Microsoft® Language シリーズ

4

Ver. 4.2~4.5

Quick BASIC

初級プログラミング入門[上]

河西朝雄[著]

技術評論社

Microsoft® Language シリーズ

4

Ver.4.2〜4.5

Quick BASIC

初級プログラミング入門[上]

河西朝雄[著]

技術評論社

はじめに

従来のBASICの大きな欠点は構造化プログラミングが行いにくいことです．しかし，BASICの持つ使いやすさ，親しみやすさ，曖昧さなどのユーザ・フレンドリな良い環境を残しながら，BASICに構造化という概念を導入すれば，そこに新しいBASICの可能性が開けるのです．

このような観点から，Quick BASICという処理系がMicrosoft社から発表されました．Quick BASICは，

- 統合開発環境
- プログラムの構造化

という2つの強力な武器を備えた新世代言語です．

本書はQuick BASICの操作法からプログラミング法を系統的にわかりやすくまとめたハンドブックで，次のような構成からなっています．

第1章　Quick BASICの概要
第2章　統合開発環境(QB)
第3章　コマンド・バージョン・ユーティリティ
第4章　Quick BASICの言語仕様
第5章　ステートメントと関数の概要
第6章　ステートメント・関数リファレンス
第7章　Ver.4.2からVer.4.5への変更点

国内でのQuick BASICは，Ver.4.2，Ver.4.5と発展してきていますが，2つのバージョンにおいて基本的には大きな違いはありません．そこで本書の本文はVer.4.2に基づいて解説し，Ver.4.2とVer.4.5の違いを第7章で説明しました．

Quick BASICは新しいBASICの道を切り開いてゆく可能性を持った言語といえます．Quick BASICを用いて，これから新しいBASICの世界へ挑戦しようとしているユーザの方々にとって本書が，多少なりともお役に立てれば幸いです．

1990年3月　河西朝雄

目　次

第3章 コマンド・バージョン・ユーティリティ

第4章　QuickBASICの言語仕様

第5章　ステートメントと関数の概要

第6章　ステートメント・関数リファレンス

第7章　Ver.4.2からVer.4.5への変更点

第1章

Quick BASIC の概要

1-1 Quick BASIC の特徴

コンピュータ言語の中からBASICをみた場合，言語仕様的には決して優れたものであるとはいえません．たとえば，ブロック構造がない，近代的流れ制御構造が完備されていない，データ型が豊富でない，モジュール化ができないなどの点です．構造化プログラミングという概念からみると，ここにBASICの限界があるといえます．

こうした状況において，Microsoft社からQuick BASICという処理系が発表されました．Quick BASICは，

1. 統合開発環境
2. プログラムの構造化

という2つの強力な武器を備えた新世代言語です．

Quick BASICは，国内では，PC-9800&AXシリーズ用のVer.4.2とバージョンアップされたVer.4.5が提供されています．

Quick BASIC Ver.4.2のおもな特徴を箇条書きにまとめると，以下のようになります。

- 高速で使い勝手のよい統合開発環境により，エディット，コンパイル，デバッグにおいて最適なプログラム開発環境が得られます．
- マウスを用いた快適なメニュー操作が可能です．
- 統合開発環境からコンパイラ/リンカを起動して，実行可能ファイル(.EXE)を作成できます．
- スタンドアロンライブラリ(.LIB)とクィックライブラリ(.QLB)という2種類のライブラリを利用することで有効な資源運用が図れます．
- コマンドバージョンのコンパイラ，リンカ，ライブラリ・マネジャをサポートします．
- MASM(マイクロソフト・マクロアセンブラ)，MS-C(Microsoft C Compiler)などのほかの言語とのインターフェースが簡単に行えます．
- ブロック概念の導入により，IF～THEN～ELSE文などをスマートに書くことができます．
- DO～LOOP，SELECT CASE文などの近代的流れ制御構造を補強し，構造的で見やすいプログラムが書けるようになりました．
- SUB，FUNCTION文で定義したサブ・ルーチンと関数は，モジュールとして扱われます．モジュール間のデータ授受は引数渡しという汎用的な方法で行い，モジュール内の変数はローカル変数として扱われるので，独立性の高いモジュールを作ることができます．

・レコード型，記号定数，配列の部分範囲指定などによりデータ構造を明確に表現することができます．
・再帰呼び出し(リカーシブ・コール)を行うことができます．
・グラフィック処理のためのステートメントが提供されています．

なお，Ver.4.5への変更点については，第7章で説明します．詳しくはそちらをごらんください．

1-2 システムの構築

1 提供されるソフトウェア

Quick BASIC Ver.4.2は，プログラム・ディスクとライブラリ&サンプルプログラムの2枚のフロッピィディスクで提供されています．また，バージョンアップしたQuick BASIC Ver.4.5は，セットアップ・ディスクとプログラム・ディスク，ライブラリ・ディスクそしてアドバイザ・ディスクの4枚のフロッピィディスクが提供されています。

●プログラム・ディスク

```
 ドライブ A: のディスク のボリュームラベルは QB42_N01
 ディレクトリは A:¥

QB_ENV        <DIR>       88-12-15    4:20 ←統合環境ディレクトリ
QB_CMP        <DIR>       88-12-15    4:20 ←コンパイラ・ディレクトリ
LIB_E         <DIR>       88-12-15    4:20 ←ライブラリ・ディレクトリ
SETUP    DOC      2763    88-12-15    4:20 ←セットアップの説明
SETUP    EXE     30198    88-12-15    4:20 ←セットアップ・ユーティリティ
PACKING  LST      3242    88-12-15    4:20 ←パッキング・リスト
README   DOC      8457    88-12-15    4:20 ←最新情報ファイル
SAMPLE   DOC      4043    88-12-15    4:20 ←サンプルプログラムの説明
        8 個のファイルがあります.     4:20
   175104 バイトが使用可能です.
```

▼統合開発に関するシステム

```
 ドライブ A: のディスク のボリュームラベルは QB42_N01
 ディレクトリは A:¥QB_ENV

.             <DIR>       88-12-15    4:20
..            <DIR>       88-12-15    4:20
MOUSE    COM      8431    88-12-15    4:20 ←マウス・ドライバ
MOUSE    SYS      8127    88-12-15    4:20 ←マウス・ドライバ
QB       BI       1783    88-12-15    4:20 ← Quick BASIC ヘッダファイル
QB       EXE    284170    88-12-15    4:20 ← Quick BASIC 本体
QB       HLP     60519    88-12-15    4:20 ←ヘルプ・ファイル
QB       INI        48    88-12-15    4:20 ←イニシャル・ファイル
QB       PIF       369    88-12-15    4:20 ← MS-WINDOWS2.0用 Pictur File
        9 個のファイルがあります.
   175104 バイトが使用可能です.
```

QB_CMP　<DIR>

▼コンパイラに関するシステム

```
 ドライブ A: のディスク のボリュームラベルは QB42_N01
 ディレクトリは A:¥QB_CMP

.             <DIR>       88-12-15    4:20
..            <DIR>       88-12-15    4:20
BC       EXE    110743    88-12-15    4:20 ← BASIC コンパイラ
BRUN42A  EXE     88539    88-12-15    4:20 ←ランタイムモジュール
BRUN42E  EXE     89328    88-12-15    4:20 ←ランタイムモジュール
LIB      EXE     35211    88-12-15    4:20 ←ライブラリ・マネジャー(ライブラリアン)
LINK     EXE     66539    88-12-15    4:20 ←リンカ
        7 個のファイルがあります.
     175104 バイトが使用可能です.
```

LIB_E　<DIR>

▼ライブラリ

```
 ドライブ A: のディスク のボリュームラベルは QB42_N01
 ディレクトリは A:¥LIB_E

.             <DIR>       88-12-15    4:20
..            <DIR>       88-12-15    4:20
BCOM42E  LIB    234477    88-12-15    4:20 ←スタンドアロン・ライブラリ
BRUN42E  LIB     25241    88-12-15    4:20 ←分離型ライブラリ
        4 個のファイルがあります.
     175104 バイトが使用可能です.
```

●ライブラリ＆サンプルプログラム・ディスク

```
 ドライブ A: のディスク のボリュームラベルは QB42_N02
 ディレクトリは A:¥

LIB_A         <DIR>       88-12-15    4:20
SOURCE        <DIR>       88-12-15    4:20
        2 個のファイルがあります.
     820224 バイトが使用可能です.
```

▼ライブラリ

```
 ドライブ A: のディスク のボリュームラベルは QB42_N02
 ディレクトリは A:¥LIB_A

.             <DIR>       88-12-15    4:20
..            <DIR>       88-12-15    4:20
BCOM42A  LIB    238895    88-12-15    4:20 ←スタンドアロン・ライブラリ
BQLB42   LIB     25301    88-12-15    4:20 ← QLB 構築ライブラリ
BRUN42A  LIB     25241    88-12-15    4:20 ←分離型ライブラリ
QB       LIB      2075    88-12-15    4:20 ←ライブラリ
QB       QLB      5784    88-12-15    4:20 ← Quick ライブラリ
        7 個のファイルがあります.
     820224 バイトが使用可能です.
```

SOURCE <DIR>

```
ドライブ A: のディスクのボリュームラベルは QB42_NO2
ディレクトリは A:¥SOURCE

.            <DIR>       88-12-15    4:20
..           <DIR>       88-12-15    4:20
BAR      BAS      6331   88-12-15    4:20
CAL      BAS      5595   88-12-15    4:20
CCALL    BAS       955   88-12-15    4:20
CCALL    C         100   88-12-15    4:20
CCALL    QLB      5351   88-12-15    4:20
COLOR    BAS      1019   88-12-15    4:20
CRLF     BAS      4441   88-12-15    4:20
CUBE     BAS      1413   88-12-15    4:20
DISPLAY  BAS      1829   88-12-15    4:20
DRAW     BAS      2376   88-12-15    4:20
EDIT     BAS      8675   88-12-15    4:20
EDIT     BI       1308   88-12-15    4:20
EDPAT    BAS      6667   88-12-15    4:20
GET      BAS       842   88-12-15    4:20
HANOI    BAS      1289   88-12-15    4:20
INDEX    BAS     10366   88-12-15    4:20
MANDEL   BAS      5560   88-12-15    4:20
PALETTE  BAS      1797   88-12-15    4:20
PSETBALL BAS      3000   88-12-15    4:20
QBL      BAS      6902   88-12-15    4:20
QLBDUMP  BAS      2783   88-12-15    4:20
SEARCH   BAS      3028   88-12-15    4:20
SIERP    BAS      2119   88-12-15    4:20
SINWAVE  BAS      1130   88-12-15    4:20
STRTONUM BAS       999   88-12-15    4:20
STYLE    BAS       387   88-12-15    4:20
TERMINAL BAS      1343   88-12-15    4:20
WHEREIS  BAS      5453   88-12-15    4:20
ABSOLUTE ASM      4520   88-12-15    4:20
INTRPT   ASM     15099   88-12-15    4:20
      32 個のファイルがあります.
  820224 バイトが使用可能です.
```

Quick BASIC Ver.4.2が提供するソフトウェアのおもなものを，機能別に分類すると以下のようになります．なお，Ver.4.5については第7章をごらんください．

表1-1　Quick BASIC 統合開発環境 QB

ファイル名	機　能
QB. EXE	Quick BASIC 本体．エディタ，インタプリタ，デバッガ内蔵
QB. HLP	ヘルプ・ファイル
QB. INI	画面の初期化情報ファイル
QB. BI	QB.QLB(Quick ライブラリ)用インクルードファイル

表1-2　BASIC コンパイラ BC

ファイル名	機　　能
BC. EXE	BASIC コンパイラ
LINK. EXE	リンカ
LIB. EXE	ライブラリ・マネジャ　(ライブラリアン)
BRUN42A. EXE	ランタイム・モジュール(代替数値演算版)
BRUN42E. EXE	ランタイム・モジュール(80?87エミュレート版)

表1-3　ライブラリ

ファイル名	機　　能	
BRUN42A. LIB	ランタイム・ライブラリ　(代替数値演算版)	コンパイラ用
BCOM42A. LIB	スタンドアロン・ライブラリ(　〃　)	
BRUN42E. LIB	ランタイム・ライブラリ　(80?87エミュレート版)	
BCOM42E. LIB	スタンドアロン・ライブラリ(　〃　)	
QB. QLB	デフォルトの Quick ライブラリ	QB 用
QB. LIB	デフォルトのライブラリ	

表1-4　その他

ファイル名	機　　能
SETUP. EXE	Quick BASIC のセットアップ・ユーティリティ
MOUSE. COM	マウス・デバイス・ドライバ(常駐/非常駐型)
MOUSE. SYS	マウス・デバイス・ドライバ(組み込み型)

2 環境の設定

■メモリの設定

Quick BASIC を動作させるためには640KB のメモリが必要です．メモリが足らなければ，PC-9801の機種に応じて適当にメモリを増設してください．

メモリ増設をした際には，MS-DOS の SWITCH コマンドを用いてメモリ・サイズの指定を行ってください．SWITCH コマンドを行ったあと，一度リセットをしてください．メモリの設定は一度しておけば，それが記憶されています．

なお，PC-9801本体のディップスイッチ SW2の No.5が ON になっていなければなりません．

```
A>switch

SWITCH  Version 2.30

RS232C-0:1200  BITS-7  PARITY-NONE  STOP-1  NONE
PRINTER :CEN24
MEMORY  :640K
NDP1    :NO
COLOR   :WHITE
BOOT    :STD

-memory[640]
```

■パスの設定

Quick BASICは，次の4つのMS-DOS環境変数を参照して処理を行います．

表1-5　MS-DOSの環境変数

環境変数	機　能
COMSPEC	再ロードするコマンド・インタプリタのパス名を設定しておく Quick BASICがOSシェル(OSに一時的に戻る)するときに，この環境変数に設定されているコマンド・インタプリタをロードする
PATH	コマンドはカレントディレクトリから探されるが，これ以外から探すときの道筋をPATH環境変数に設定する
LIB	ライブラリの検索パスを設定する
TMP	LINKが作業に使うテンポラリファイルを置く位置を設定する RAMディスクがあれば，そこに置くようにすると高速処理が行える

■日本語FEPの組み込み

Quick BASICで使用できる日本語FEP(フロント・エンド・プロセッサ)は次の2種類です．

- **ATOK6またはATOK7**
- **MS-KANJI　API仕様の日本語入力システム**

ここではATOK6を組み込む方法を説明します．ATOK6を使用するには，次のファイルが必要です．

ATOK6A.SYS ─┐ **かな漢字変換デバイス・ドライバ**
ATOK6B.SYS ─┘
ATOK.DIC ── **辞書ファイル**

Quick BASICのインタプリタ/コンパイラシステムを1MBフロッピィにセットアップした場合，ディスク容量がわずかしか残らないため，日本語システムはドライブBにのせなければなりません．この場合のCONFIG.SYSでの設定は，次のように記述します．

● CONFIG. SYS内の記述

```
DEVICE＝B：ATOK6A. SYS /D＝B
DEVICE＝B：ATOK6B. SYS
```

■マウス・ドライバの組み込み

Quick BASICでは次の2つのマウス・ドライバを提供しています．

表1-6　マウス・ドライバの種類

マウス・ドライバ	機　能
MOUSE. SYS	従来のデバイス・ドライバで，CONFIG.SYSの中でDEVICE＝MOUSE.SYSと指定することでシステム起動時に組み込まれる．起動後はマウス・ドライバをシステムから切り離すことはできない
MOUSE. COM	コマンドラインからコマンドを投入することにより，マウス・ドライバを常駐/非常駐させることができる．コマンド書式は次のとおりである MOUSE　バス・マウスを利用する MOUSE /C　シリアル・マウスを利用する MOUSE OFF　MOUSE. COMをメモリから解放する したがってこのMOUSE.COMはAUTOEXEC. BATの中でシステム起動時に自動実行させて組み込む

SETUPユーティリティではどちらのマウス・ドライバを使用するのか問い合わせてきますので，MOUSE.SYSまたはMOUSE.COMを選択します．

MOUSE.COMのほうが，常駐/非常駐の制御をユーザが行える点で便利だと思います．

■プリンタ・ドライバの組み込み

従来のMS-DOSでは，プリンタ・デバイス・ドライバはMS-DOSシステムの中にあらかじめ組み込まれていました．ところが，MS-DOS Ver.3.3とMS-DOS Ver.3.1の一部(製品番号PS98-011-×××)では，PRINT.SYSという専用のプリンタ・デバイス・ドライバとしてMS-DOSから分離しています．したがってこの場合は，ユーザがCONFIG.SYSの中でPRINT.SYSを登録しなければなりません．

● CONFIG. SYS内の記述(MS-DOS Ver.3.1の一部とVer.3.3)

```
DEVICE＝PRINT. SYS
```

■ライブラリの選択

Quick BASICのコンパイラ(BC.EXE)/リンカ(LINK.EXE)で利用するライブラリは,

- **生成する実行可能ファイルのタイプ**
- **数値演算コ・プロセッサ(8087/80287)の使用の有無**

の違いにより，それぞれ次のように異なるライブラリを使います.

表1-7　コンパイラ用ライブラリの種類

コ・プロセッサ ＼ EXEのタイプ	代替数値演算	8087/80287エミュレート
スタンドアロン型	BCOM42A. LIB	BCOM42E. LIB
ランタイム分離型	BRUN42A. EXE BRUN42A. LIB	BRUN42E. EXE BRUN42E. LIB

スタンドアロン型のEXEファイルは単独で動作しますが，ランタイム分離型のEXEファイルは実行時にランタイム・モジュール(BRUN42A.EXEまたはBRUN42E.EXE)をロードし，このモジュールの協力下において動作します.

代替数値演算ライブラリは，数値演算コ・プロセッサを使用せずに数値演算を行います. 8087/80287エミュレート・ライブラリは，数値演算コ・プロセッサを使用して数値演算を行います.

統合開発環境QBでは一般にライブラリを必要としませんが，QB.QLB，QB.LIBなどのライブラリを使用することもできます.

3 システムの構築例

■ SETUPユーティリティ

QuicK BASICはSETUPユーティリティを用いて簡単にフロッピィディスクまたはハードディスクにシステムを構築できます. ただし，MS-DOSシステムをあらかじめコピーしておきます.

```
A>SETUP
```

```
ＱｕｉｃｋＢＡＳＩＣ　セットアップ　ユーティリティ　　　　Ｖｅｒ１．００

---準備---                                                          (1/3)
◎フロッピーディスクへセットアップする場合、フォーマット済みのディスクを用意し
　てください．

　［フロッピーベースでＱｕｉｃｋＢＡＳＩＣを利用する方へ］
　あらかじめフォーマット済みのディスクが必要です．
　メディアのタイプによって必要なディスクの枚数が異なります．
　ワークディスクには FORMAT /S スイッチでＭＳ－ＤＯＳシステムを転送しておいて
　下さい．システムが転送されていない場合はセットアップはおこなわれません．
　・ＨＤタイプ(3.5 2HD, 5 2HD)の場合
　　　　１枚：ワークディスク
　・ＤＤタイプ(3.5 2DD, 5 2HD)の場合
　　　　２枚：ワークディスク、コンパイラディスク
　　　　(FORMAT /9 で 720Kフォーマットにしておく方が良いでしょう．)
◎フォーマットが済みましたら添付のシール(「ワークディスク」，「コンパイラディ
　スク」)をディスクに貼っておいて下さい．他のシールは使いません．御自由にお使い
　下さい．
フォーマット済みのディスクが用意していなければ，CTRL+C でセットアップを中止して
ディスクのフォーマットを行って下さい．

【何かキーを押して下さい】←何かキーを押すと次の画面に進む．
```

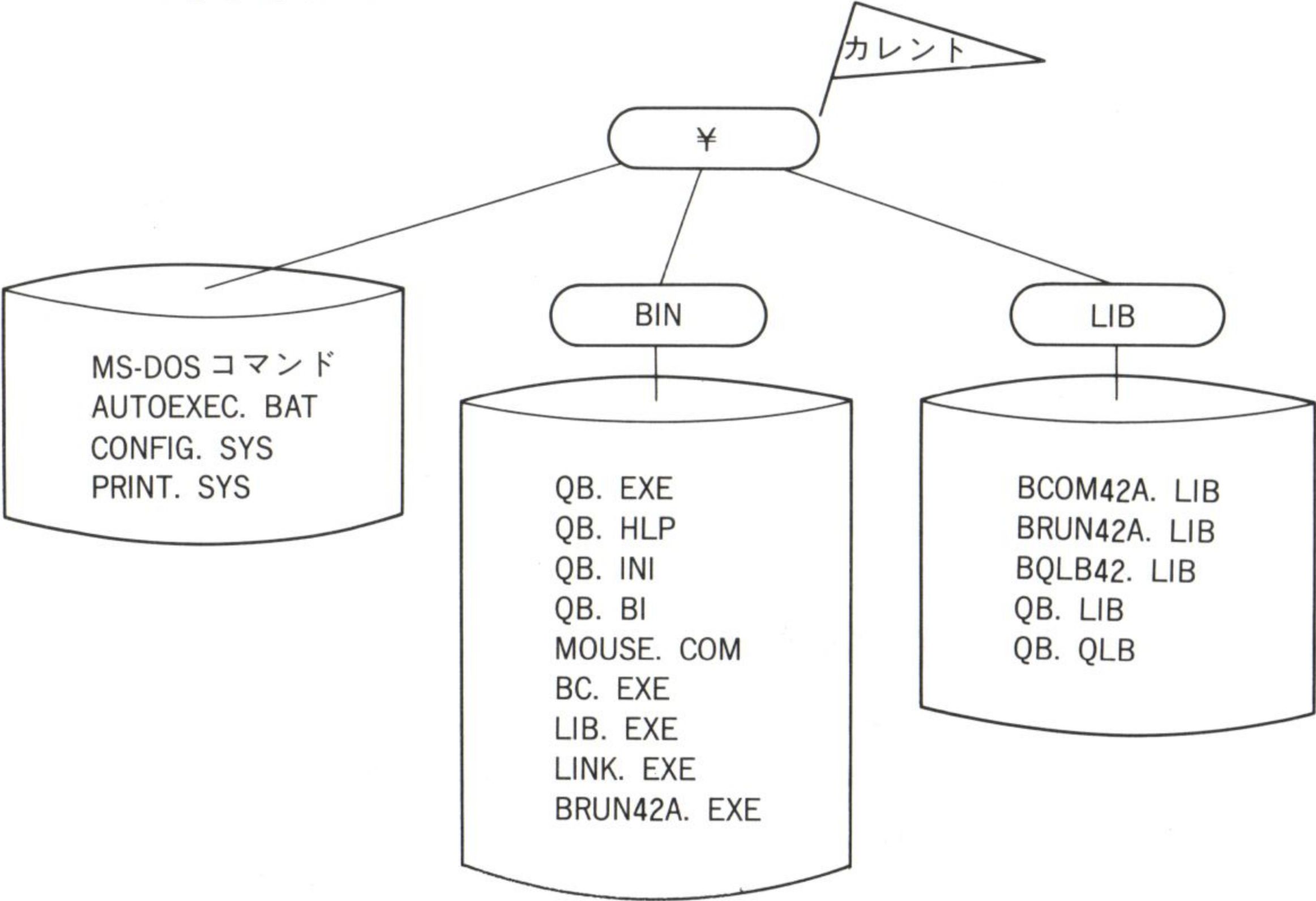
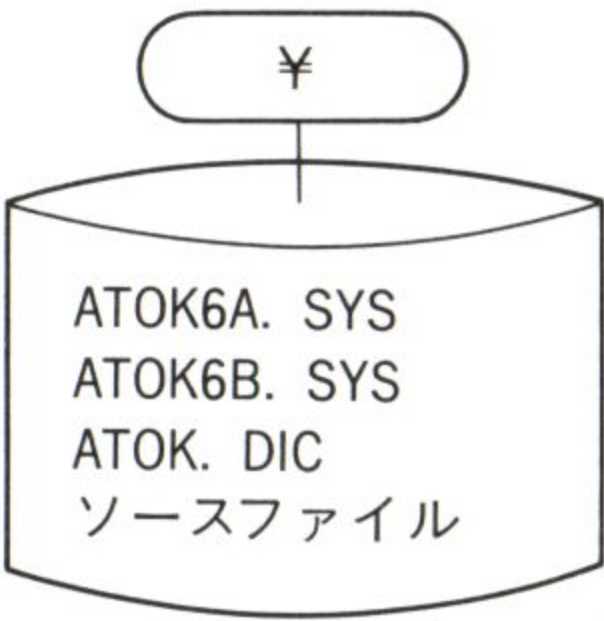
```
ＱｕｉｃｋＢＡＳＩＣ　セットアップ　ユーティリティ　　　　Ｖｅｒ１．００

---サンプルプログラム---                                            (2/3)
セットアップでは、サンプルプログラムを転送しません．必要に応じて転送をおこなっ
て下さい．

---ライブラリについて---
ＱｕｉｃｋＢＡＳＩＣのコンパイラ(BC.EXE)が通常利用するライブラリは、以下のもの
です．
BRUN42A.EXE　ランタイムモジュール
BRUN42A.LIB　ランタイムライブラリ
BCOM42A.LIB　スタンドアロンライブラリ

これらを代替数値演算ライブラリ(ALT-MATH)と呼びます．これらは、数値演算コプロセ
ッサの有無に関わらず、一定の速度で高精度な計算を実現します．
数値演算コプロセッサを装着している方のため BRUN42E.EXE,BRUN42E.LIB,BCOM42E.LIB
(80?87エミュレートライブラリ)を用意しています．これによってより高度な演算を実現
できます．
フロッピーベースでご利用になる場合は、セットアップではエミュレートライブラリは
転送できません．必要な場合は、マスターディスクから転送して下さい．

【何かキーを押して下さい】←何かキーを押すと次の画面へ進む
```

```
ＱｕｉｃｋＢＡＳＩＣ　セットアップ　ユーティリティ　　　Ｖｅｒ１．００

---フロントプロセッサの組み込み---
                                                              (3/3)
ＱｕｉｃｋＢＡＳＩＣに日本語入力システムを組み込む場合は、お使いのフロントプロ
セッサのマニュアルを参照して各自おこなってください．
なお、ご利用可能なフロントプロセッサは、ＡＴＯＫ６、ＭＳ－ＫＡＮＪＩ ＡＰＩの
ものです．添付の「日本語入力システムのご使用について」を参考にしてください．

---ＣＯＮＦＩＧ．ＳＹＳ、ＡＵＴＯＥＸＥＣ．ＢＡＴ---

このセットアッププログラムは、フロッピーへセットアップしたとき「ワークディスク」
に CONFIG.SYS , AUTOEXEC.BAT を作成します．ハードディスクの場合は、指定されたデ
ィレクトリに、NEW-CONF.SYS, NEW-PATH.BAT を作成しますので，これらのファイルを参
考にして CONFIG.SYS AUTOEXEC.BAT を修正してください．環境変数については、
README.DOC をご参照下さい．

なお、README.DOC には補足の情報や，訂正資料が記載されています．必ずお読み頂くよ
うお願いします．

【何かキーを押して下さい】 ←何かキーを押すと次の画面に進む.
```

```
ＱｕｉｃｋＢＡＳＩＣ　セットアップ　ユーティリティ　　　Ｖｅｒ１．００

●ハードディスクにセットアップしますか [Y/N] ？ Ｎ
●ＱｕｉｃｋＢＡＳＩＣをどのドライブにインストールしますか？ Ｂ
●マウスを使用しますか [Y/N] ？ Ｙ                                    ─各項目に答える
●マウスの種類を選択して下さい [1:ﾊﾞｽ/2:ｼﾘｱﾙ] ？ １
●マウスドライバを選択して下さい [1:MOUSE.SYS/2:MOUSE.COM] ？ ２


上記の設定でよろしいですか　[Y/N]？Ｙ ←Yと答えるとSETUP作業が開始される
--------------------------------------------------------------------------------
マウスをサポートするデバイスドライバの種類を選択して下さい．
メモリ内に常駐するタイプが１（ＭＯＵＳＥ．ＳＹＳ）で、必要に応じて組込み
が可能なタイプが２（ＭＯＵＳＥ．ＣＯＭ）です．
ＭＯＵＳＥ．ＣＯＭは、ＭＯＵＳＥ　ＯＦＦ　でメモリ解放ができる為、ＣＯＭ
形式を使うことをお勧めします．
```

■1MB フロッピィでの構築例

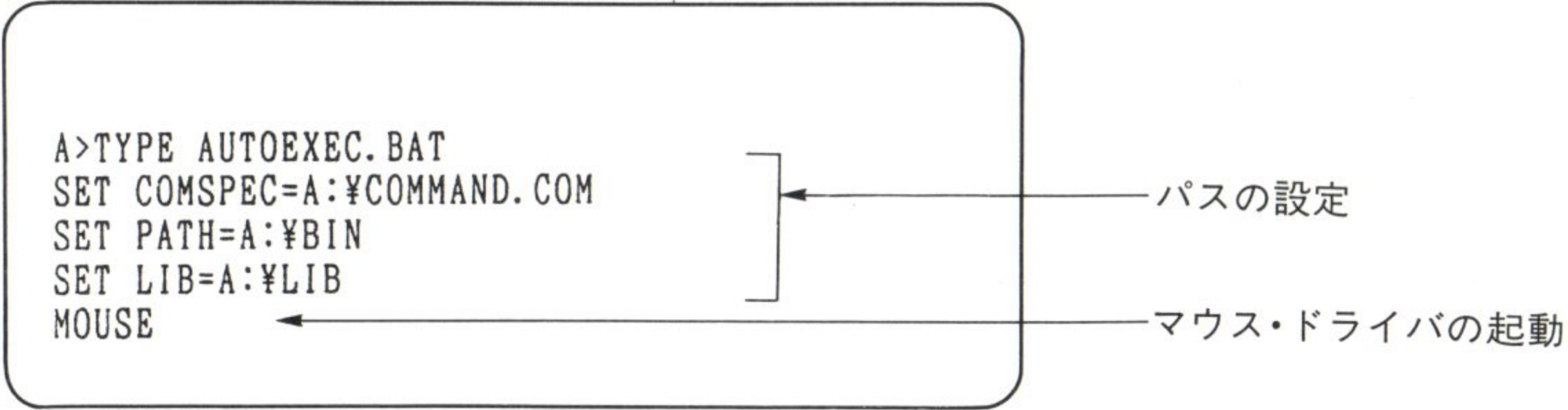
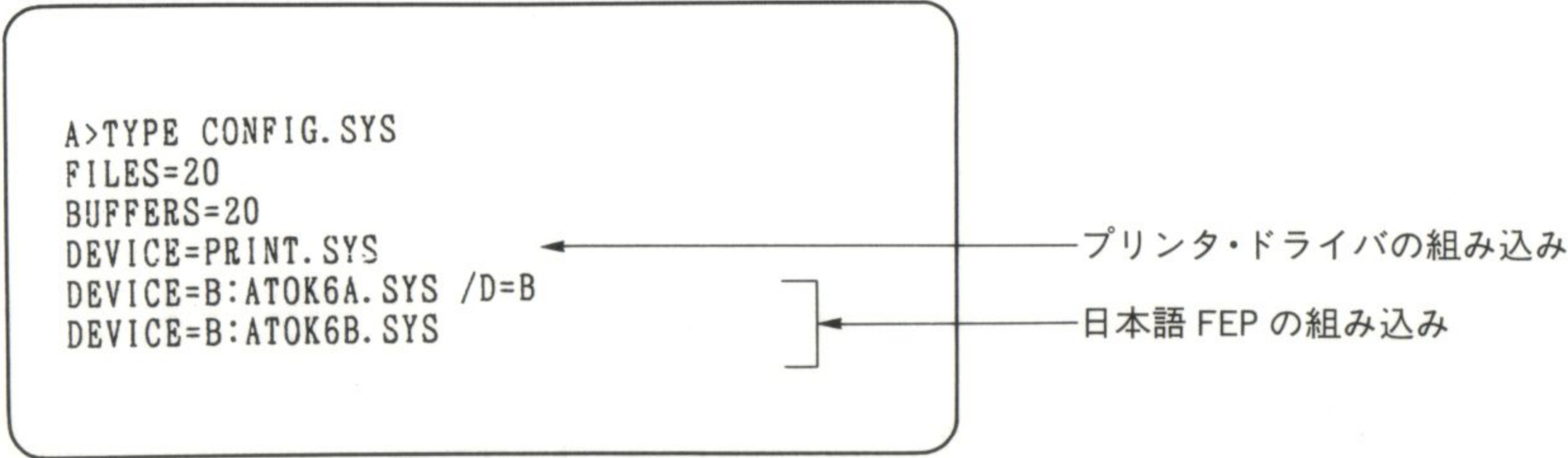
日本語辞書をドライブＢに置きます．
マウス・ドライバはMOUSE.COMを選択します．
ソースファイルはドライブＢに作っていきます．

● ドライブ A

● ドライブ B

このようなシステム環境では，AUTOEXEC.BAT と CONFIG.SYS を次のように作ります．

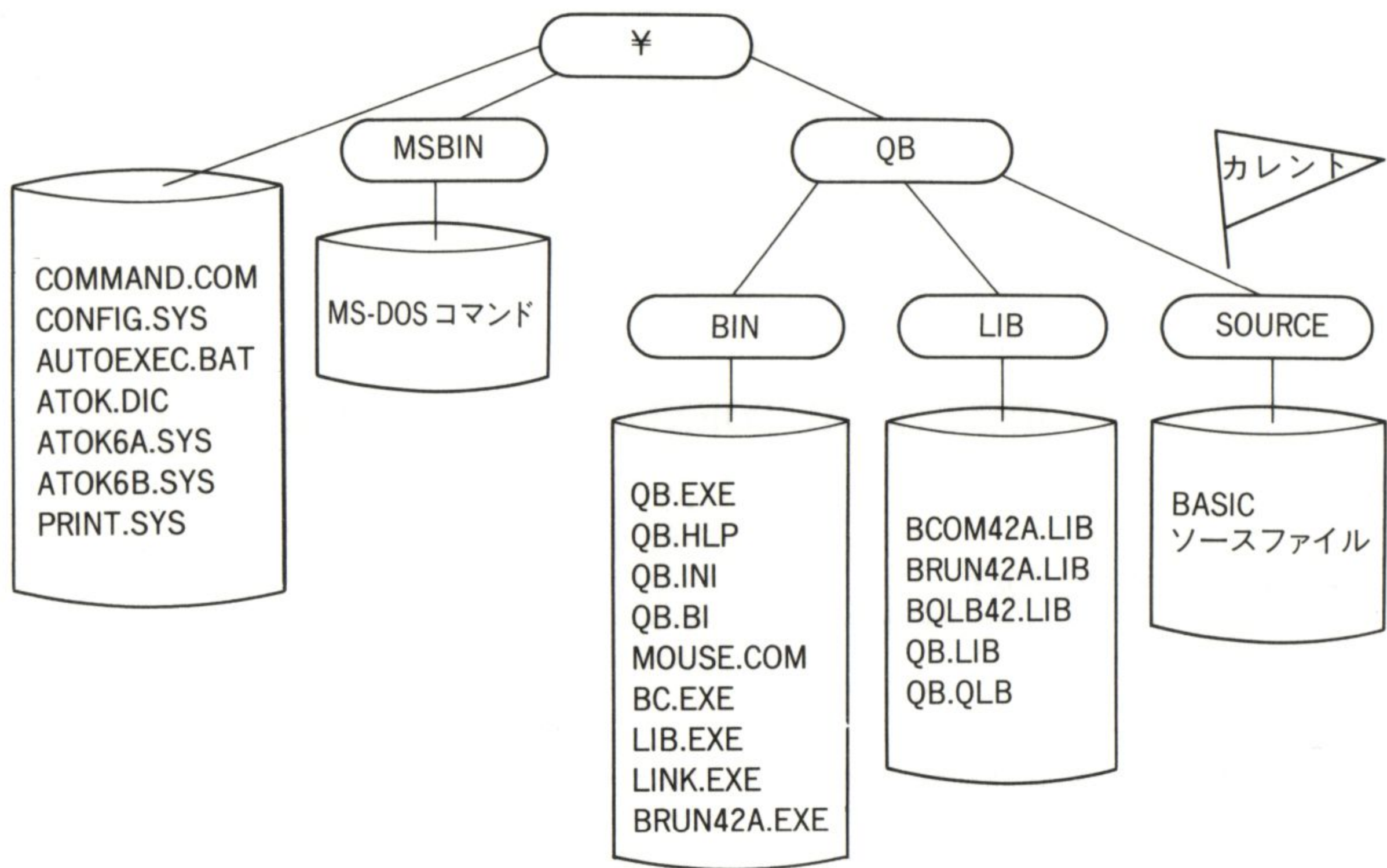
● AUTOEXEC. BAT

```
A>TYPE AUTOEXEC.BAT
SET COMSPEC=A:¥COMMAND.COM
SET PATH=A:¥BIN
SET LIB=A:¥LIB
MOUSE
```

● CONFIG. SYS

```
A>TYPE CONFIG.SYS
FILES=20
BUFFERS=20
DEVICE=PRINT.SYS                    ←プリンタ・ドライバの組み込み
DEVICE=B:ATOK6A.SYS /D=B            ┐
DEVICE=B:ATOK6B.SYS                 ┘←日本語 FEP の組み込み
```

注）BASIC コンパイラとライブラリを除いて QB システムだけにすれば，ATOK 関係をドライブ A にのせることができます．

■ハードディスクでの構築例

ハードディスクのような大容量システムでは，Quick BASIC 以外の言語もディスク上にいっしょにのせることが考えられるので，Quick BASIC のシステムは，QB というサブディレクトリを設け，その下に置くことにします．また，ユーザファイルもハードディスク上に置いたほうが有利なので，これを SOURCE というサブディレクトリに置き，ここをカレントディレクトリにします．このような点を考慮したハードディスクでのシステム構築例を，以下に示します．

このようなシステム環境では，AUTOEXEC.BAT と CONFIG.SYS を次のように作ります．

● AUTOEXEC. BAT

```
SET COMSPEC=¥COMMAND.COM
SET PATH=¥QB¥BIN;¥MSBIN;¥
SET LIB=¥QB¥LIB
MOUSE
CD ¥QB¥SOURCE
```

● CONFIG. SYS

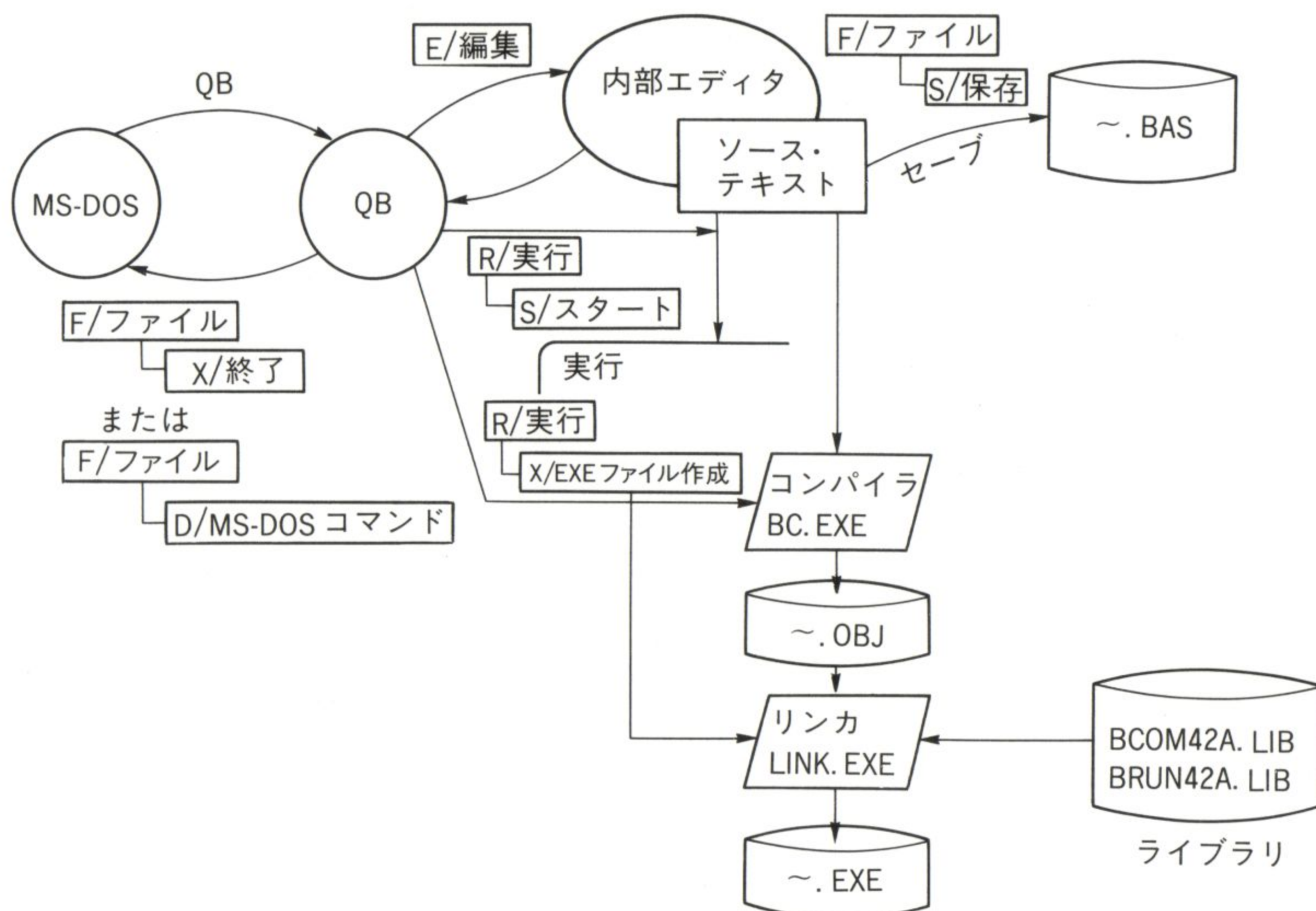
```
FILES=20
BUFFERS=20
DEVICE=ATOK6A. SYS
DEVICE=ATOK6B. SYS
DEVICE=PRINT. SYS
```

1-3 Quick BASICの運用形態

提供されるQuick BASICのシステム・ディスクの中には，統合開発環境QBとBASICコンパイラBCの2つのシステムが入っています．

統合開発環境はQuick BASICの基本システムで，QB.EXEを本体とし，この中にエディタ，インタプリタ，デバッガが内蔵されています．一度QBを起動すれば，プル・ダウン・メニュー方式で各処理を高速に行うことができます．

QBは基本的にはインタプリタです．しかしプル・ダウン・メニューからBASICコンパイラ(BC.EXE)とリンカ(LINK.EXE)を自動的に呼び出し，実行可能ファイル(.EXE)を生成することもできます．

BC.EXE，LINK.EXEは従来のコンパイラ処理系と同様に，コマンドラインからコンパイル/リンク処理を行うこともできます．

第2章

統合開発環境（ＱＢ）

2-1 QB の起動とメニュー画面

QB の起動は,

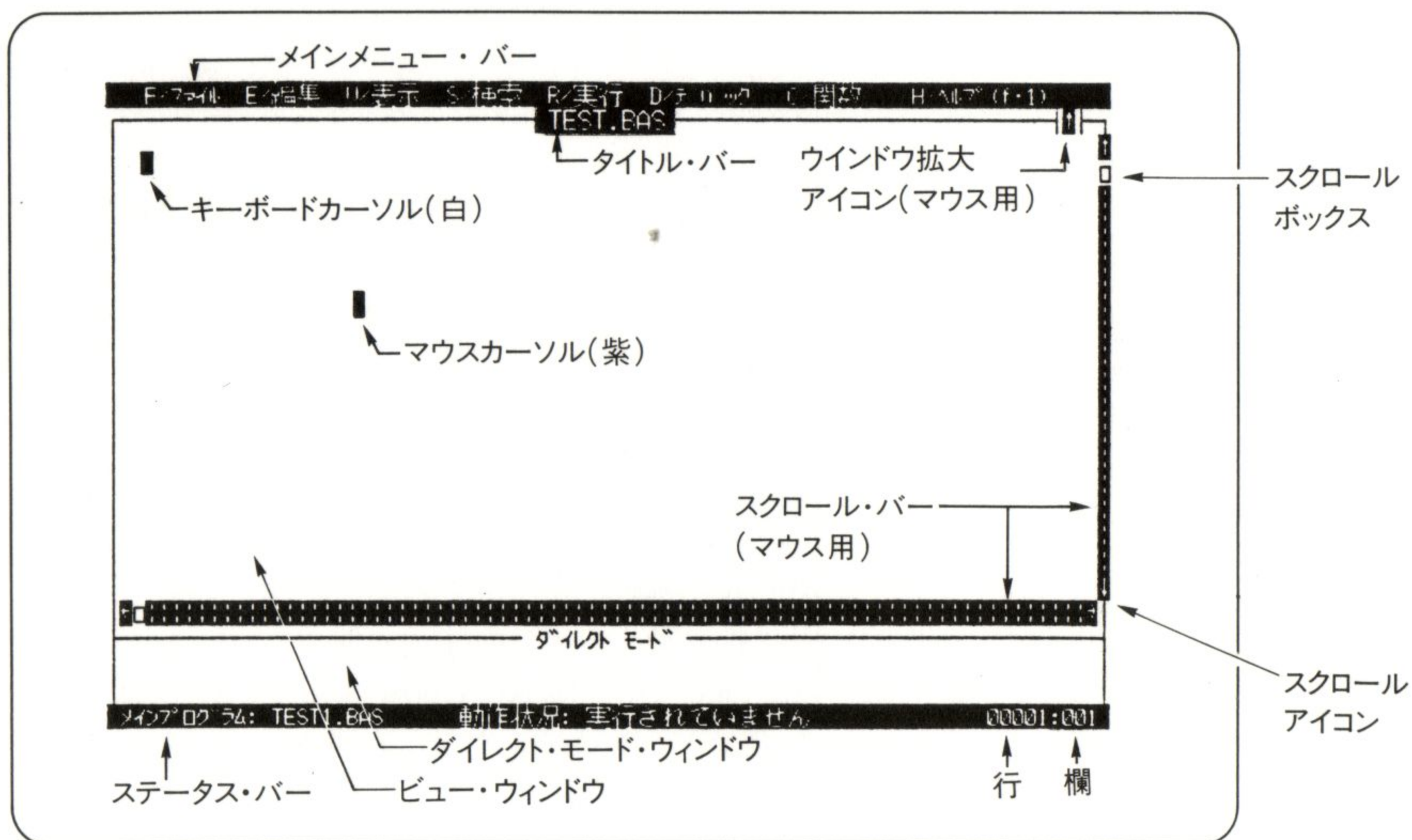
A>QB ↵

と単独で起動し，ファイルセーブ時にファイル名を指定するか,

A>QB␣B：TEST ↵

└ソースファイル名．ファイルタイプを省略すると.BAS とみなされる．

とソースファイル名を指定して起動します．
これにより，次のようなメニュー画面が表示されます．

メインメニュー・バー
TEST.BAS
タイトル・バー
ウインドウ拡大アイコン(マウス用)
スクロールボックス
キーボードカーソル(白)
マウスカーソル(紫)
スクロール・バー(マウス用)
ダイレクト モード
スクロールアイコン
メインプログラム: TEST1.BAS 動作状況: 実行されていません 00001:001
ダイレクト・モード・ウィンドウ
ステータス・バー
ビュー・ウィンドウ
行
欄

■ QBの起動オプション

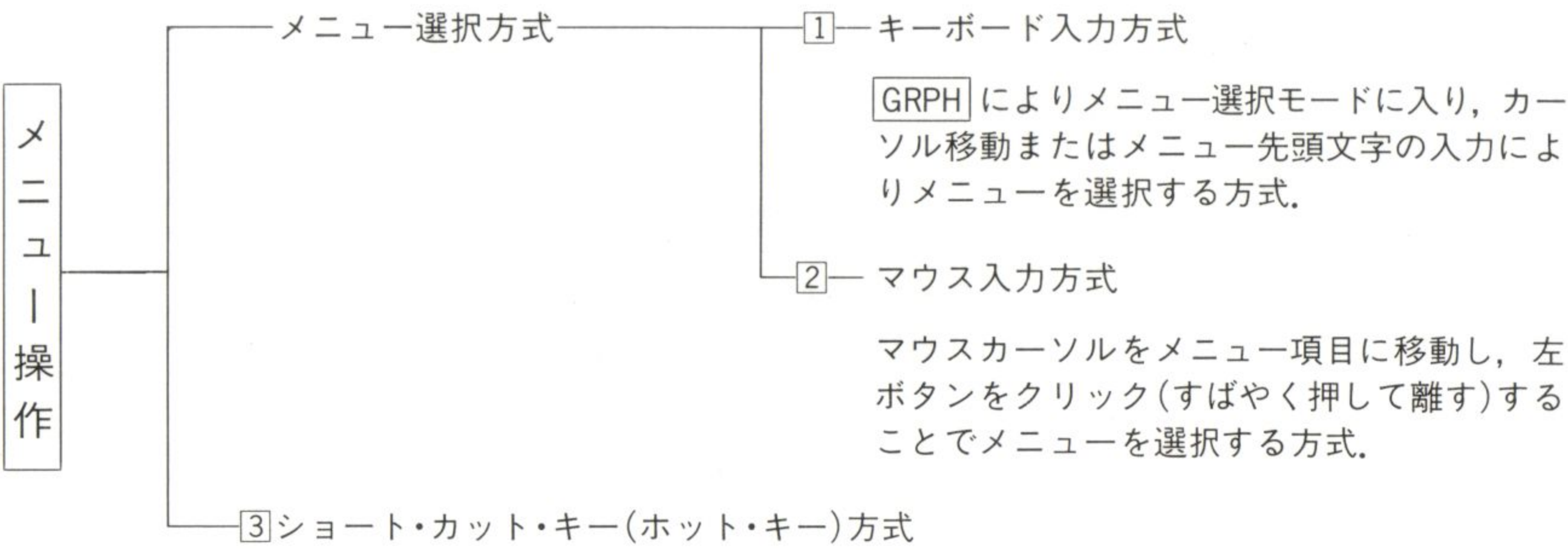
QBの起動時に，次のようなオプションを指定することができます．オプション文字の大/小は区別されません．

オプション	機　　能
/AH	動的配列に対するメモリの解放(第3章3-1の2参照)
/B	モノクロ画面表示にする
/C:n	通信ポート・バッファのサイズの指定(第3章3-1の2参照)
/CMD string	string で示される文字列を COMMAND$関数に引き渡す．このオプションはコマンドラインの最後に指定する
/L[qlib]	qlib で示されるクイックライブラリを QB システムに読み込む．qlib を省略すると QB.QLB が読み込まれる
/MBF	IEEE 形式の数値をマイクロソフトバイナリ形式の数値として扱う(第3章3-1の2参照)
/NOHI	2階調表示のディスプレイを使うときに/B オプションと組み合わせて指定する
/RUN sourcefile	sourcefile で示されるソースファイルを QB 起動時に読み込み，実行する

2−2 Quickメニューの使い方

1 メニュー操作の基本

■メニュー操作の方式

メニュー操作は，おおよそ次の3つの方法のいずれかで行うことができます．

メニュー操作
- メニュー選択方式
  - 1 キーボード入力方式
    GRPH によりメニュー選択モードに入り，カーソル移動またはメニュー先頭文字の入力によりメニューを選択する方式．
  - 2 マウス入力方式
    マウスカーソルをメニュー項目に移動し，左ボタンをクリック(すばやく押して離す)することでメニューを選択する方式．
- 3 ショート・カット・キー(ホット・キー)方式
  メニューの一部のものはファンクションキー(CTRL, SHIFT と組み合わせる場合もある) に割り当てられているので，これを1タッチすることでメニューを選択する方式．

■メニュー操作で使うキー

メニュー操作で使うおもなキーは以下のものです.

表2-1　メニュー操作に必要なキー

メニュー操作キー	機　　能
GRPH	メニュー選択モードに入る.メニュー項目の先頭文字の色が変わる メニュー選択モードから抜けるには再び GRPH を押す メインメニューの選択では GRPH を押し続ける必要はないが,サブメニュー内では押しながらメニュー選択を行う必要がある場合がある
TAB	サブメニュー内の項目を次に進める
SHIFT + TAB	サブメニュー内の項目を1つ前に戻す
↵	メニューバー(カーソル)の置かれているメニューを実行する
← →	メインメニュー・バーを移動する
↑ ↓	サブメニュー・バーを移動する
ESC	メニュー処理を取り消す

■マウス入力方式

●メニューの選択

マウス・カーソルをメニューに合わせ左ボタンをクリック(マウスボタンをすばやく押して離すこと)します.

メインメニューを取り消すには,メニュー枠の外にカーソルを移し,左ボタンをクリックします.

サブメニューを取り消すには,取消にカーソルを移し,左ボタンをクリックします.

●コマンドの選択

マウス操作	機　　能
ウィンドウのクリック	ウィンドウをアクティブにする
カーソルをタイトルバーに移し，タイトルバーを上下に移動	アクティブウィンドウの拡大/縮小
拡大アイコン(|↑|)をクリックするか，タイトルバーをダブルクリック	アクティブウィンドウを画面一杯に拡大する
スクロールボックス（□）にカーソルを移し，スクロールバー上で目的位置まで左ボタンを押しながら移動	テキストのスクロール
スクロールバー上の□で分離された上半分または下半分のクリック	1ページ単位の上/下スクロール
スクロールアイコン(↑，↓，←，→)をクリック	1行または1文字単位の上/下，左/右のスクロール

■サブメニューの意味

サブメニューの各項目の一般的な意味を示します．

メインメニューのメニュー・バー

F/ファイル　E/編集　V/表示　S/検索

U/元に戻す　GRPH+BS
T/カット　SHIFT+DEL
C/コピー　CTRL+INS
P/ペースト　SHIFT+INS
E/削除　DEL
S/新規SUB...
F/新規FUNCTION...
>Y/構文チェック

サブメニューのメニュー・バー

対応するショート・カット・キー

先頭の文字の色が変わっていないものは，今の状態では実行できないことを示す

...があるサブメニューは，この下にさらにサブメニューがあることを示す

>があると，この項目が示す機能がON状態にあることを示す
この項目を選択するたびにON/OFFのトグル動作となる

注）Ver. 4.5では，>ではなく＊の表示

2 Quick メニュー一覧

Quick BASIC のメインメニューとサブメニューの一覧を次に示します.

表2-2　メニュー一覧

メインメニュー	サブメニュー	機能	ショート・カット・キー	解説項
F/ファイル	N/新規	メモリ上のプログラムを消去する		2-4の2
	O/読込...	メモリ上に新しいプログラムをロードする		2-4の3
	M/追加...	メモリ上に別のプログラムを追加ロードする		〃
	S/保存	アクティブウィンドウのプログラムをセーブする		2-4の1
	A/名前を変えて保存...	アクティブウィンドウのプログラムを別のファイル名,別の保存モードでセーブする		〃
	V/すべて保存	メモリ上のすべてのモジュールをセーブする		〃
	C/サブファイル作成...	新しいモジュール/インクルードファイル/ドキュメントファイルを作成する		2-4の5 6,7
	L/サブファイル常駐...	モジュール/インクルードファイル/ドキュメントファイルをメモリ上にロードする		〃
	U/サブファイル解放...	メモリ上のモジュール/インクルードファイル/ドキュメントファイルを消去する		〃
	P/印刷...	メモリ上のプログラムをプリンタに印刷する		2-4の4
	D/MS-DOS コマンド	一時的に MS-DOS に抜ける.OS シェル		2-9の2
	X/終了	QB モードを終了して MS-DOS に戻る		2-9の1
E/編集	U/元に戻す	最後の編集動作を行う前の状態に戻す アンドゥ機能	GRPH + BS	2-5の9
	T/カット	指定されたテキストを削除し,クリップボードに格納する	SHIFT + DEL	2-5の6
	C/コピー	指定されたテキストをクリップボードに格納する	CTRL + INS	〃
	P/ペースト	クリップボードの内容を,現在のカーソル位置に挿入する	SHIFT + INS	〃
	E/削除	指定されたテキストを削除する	DEL	2-5の5
	S/新規 SUB...	新しい SUB プロシージャのウィンドウを開く		2-5の9
	F/新規 FUNCTION...	新しい FUNCTION プロシージャのウィンドウを開く		〃
	Y/構文チェック	自動構文チェック機能の ON/OFF を切り換える		2-5の3

メインメニュー	サブメニュー	機　　　　能	ショート・カット・キー	解説項
V/表示	S/SUB 一覧...	メモリ上の SUB プロシージャの一覧を表示する	F・2	2-4の5
	E/次の SUB	アクティブウィンドウに, 次のサブプログラムを表示	SHIFT + F・2	2-4の5
	P/画面分割	画面分割と単一ウィンドウを切り換える		2-3の1
	N/次のステートメント	次に実行されるステートメントを表示する		2-7の7
	U/実行画面	出力画面に切り換える	F・4	2-3の4
	I/インクルードファイル編集	インクルードファイルを表示して編集モードに入る		2-4の7
	L/インクルードファイル表示	インクルードファイルを表示する		〃
	O/オプション...	画面の表示色, タブ・サイズの設定をする		〃
S/検索	F/検索...	指定したテキストを探す		2-5の7
	S/指定文字列検索	アクティブウィンドウ中で反転表示されているテキストを検索する	CTRL + ¥	〃
	R/次を検索	次に一致する検索文字列を探す	F・3	〃
	C/置換...	指定したテキストを探し新しいテキストに置き換える		〃
	L/ラベル...	ラベルの検索を行う		
R/実行	S/スタート	プログラムを実行する	SHIFT + F・5	2-6の1
	R/リスタート	プログラム中の変数をクリアし, プログラム先頭を実行開始点に設定する		2-7の7
	N/続行	STOP 文などにより中断されていたプログラムを続行する	F・5	〃
	C/COMMAND$入力...	COMMAND$関数が返す文字列を設定する		2-6の4
	X/EXE ファイル作成...	コンパイラ/リンカを起動し, 実行可能ファイル (.EXE) をディスクに作成する		2-6の2
	L/ライブラリ作成...	コンパイラ/リンカ/ライブラリを起動し, クィック・ライブラリをディスクに作成する		2-6の3
	M/メインモジュールの設定...	実行の開始点になるメインモジュールを他へ移す		
D/デバッグ	A/ウォッチの追加...	ウォッチ式を設定する		2-7の3
	W/ウォッチポイント...	ウォッチポイントを設定する		2-7の5
	D/ウォッチの削除...	ウォッチウィンドウ内のウォッチ式を削除する		2-7の3
	L/全ウォッチの削除	ウォッチウィンドウ内の全ウォッチ式を削除する		〃
	T/トレース	トレースモードを ON/OFF する		2-7の2

メイン メニュー	サブメニュー	機　　　能	ショート・カット・キー	解説項
D/デバッグ	H/ヒストリ	ヒストリ機能を ON/OFF する		2-7の8
	B/ブレークポイント	カーソル位置にブレークポイントを設定/解除する	F・9	2-7の4
	C/全ブレークポイントの解放	すべてのブレークポイントを解除する		〃
	S/カレントステートメントの設定	次に実行するステートメントをカーソル位置に移す		2-7の7
C/関数		現在呼び出されているプロシージャを表示し，そのプロシージャを呼び出しているプロシージャ名の位置にカーソルを移す		2-7の9
H/ヘルプ	G/全般...	全般のヘルプ情報を表示する	F・1	2-8の2
	T/キーワード	BASIC のキーワード(予約語)についてのヘルプ情報を表示する	SHIFT + F・1	2-8の1
	C/ヘルプをとじる	ヘルプウィンドウを閉じる	ESC	2-8の2

3 ショート・カット・キー

Quick BASICのショート・カット・キーの一覧を以下に示します.テキストの編集に関するものは2-5を参照してください.

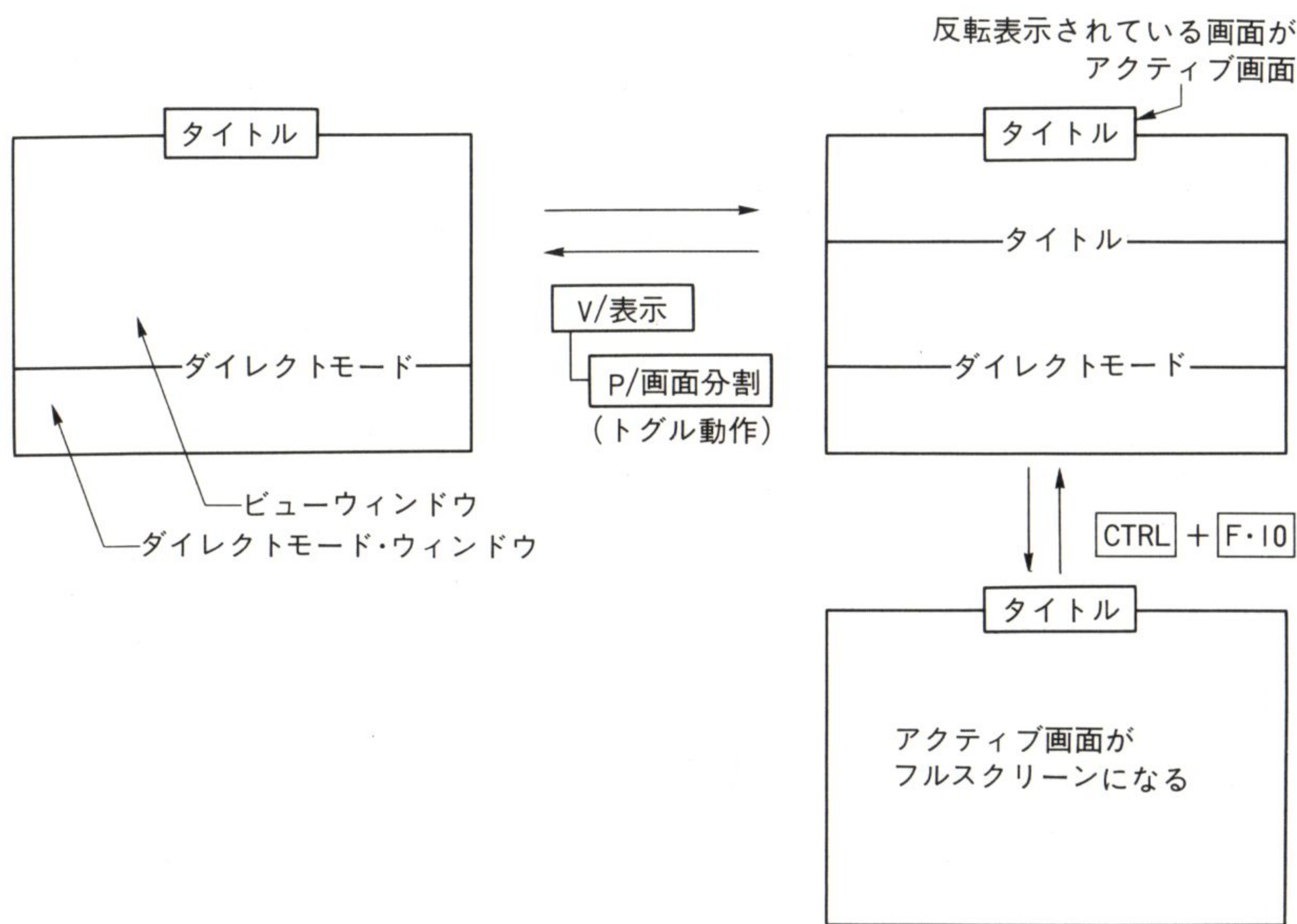
表2-3　ショート・カット・キー

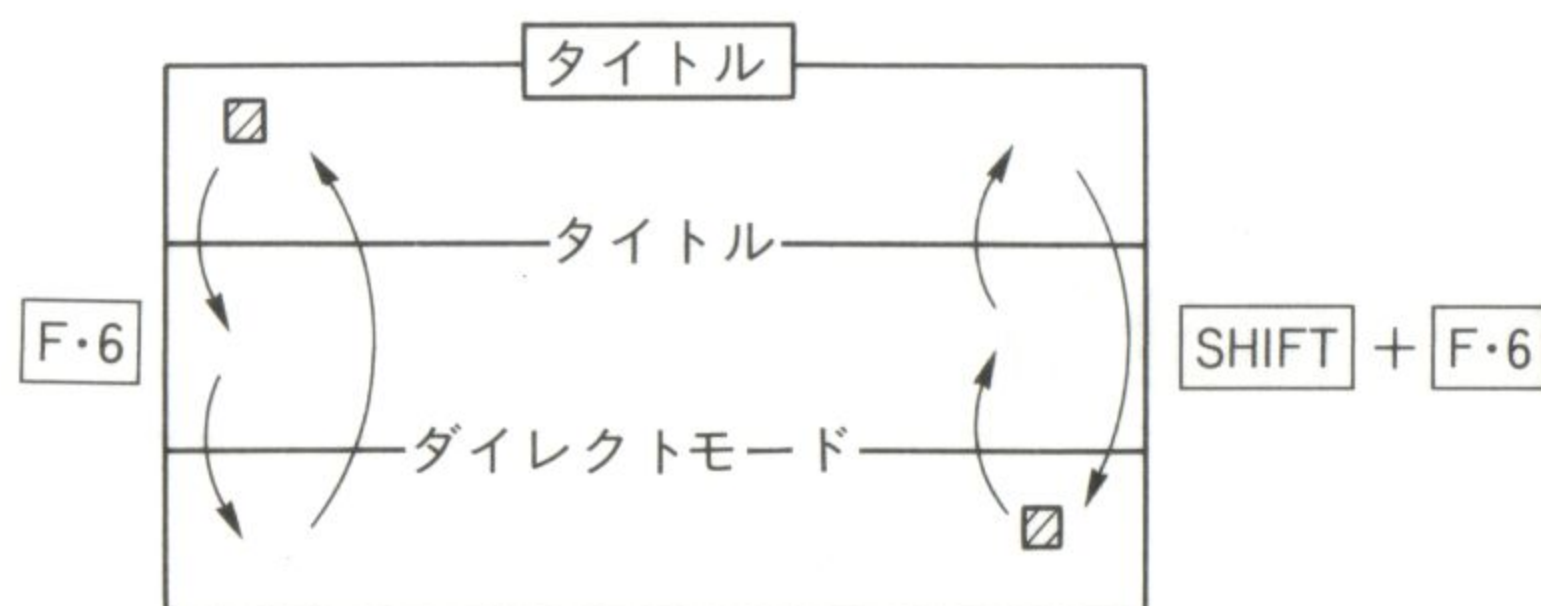
メニュー操作キー	機　　能
F・1	全般のヘルプ情報を表示する
F・2	SUB モジュールの一覧を表示する
F・4	QB 画面とプログラム出力画面を切り換える
F・5	STOP 文などにより中断されていたプログラムを続行する
F・6	アクティブ・ウィンドウを次の画面(下方の画面)に移す
F・7	現在のカーソル位置までプログラムを実行する
F・8	1行トレース.プロシージャの内部もトレースする
F・9	カーソル行に,ブレークポイントを設定/解除する
F・10	1行トレース.プロシージャの内部はトレースしない
SHIFT + F・1	指定された予約語(キーワード)についてのヘルプ情報を表示する オンライン・ヘルプ
SHIFT + F・2	アクティブ・ウィンドウに次の SUB プロシージャを表示する
SHIFT + F・5	プログラムを最初から実行する
SHIFT + F・6	アクティブ・ウィンドウを前の画面(上方の画面)に移す
SHIFT + F・8	実行された最後の20行をプログラム先頭方向にたどる
SHIFT + F・10	実行された最後の20行をプログラム終末方向にたどる
CTRL + F・2	アクティブ・ウィンドウに前の SUB プロシージャを表示する
CTRL + F・5	フルスクリーンを以前の分割画面に戻す
CTRL + F・10	フルスクリーンと分割画面を切り換えるトグル動作

2−3 ウィンドウ操作

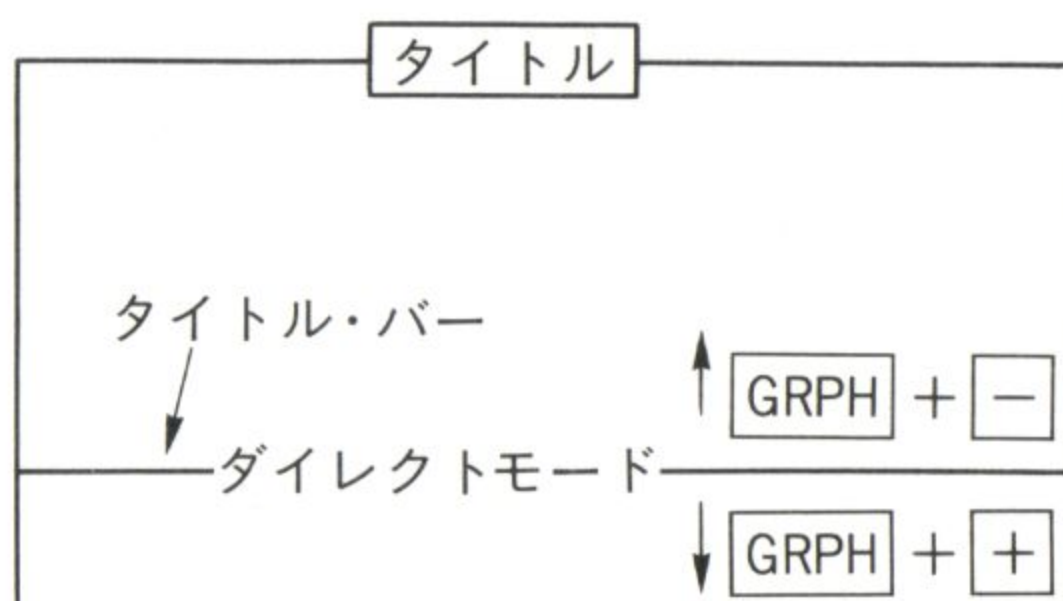
1 ウィンドウの分割

ウィンドウは，ビューウィンドウ，ダイレクトモード・ウィンドウ，ウォッチ･ウィンドウに分かれます．このうちビューウィンドウは，[V/表示]—[P/画面分割]メニューで2つに分割することができます．

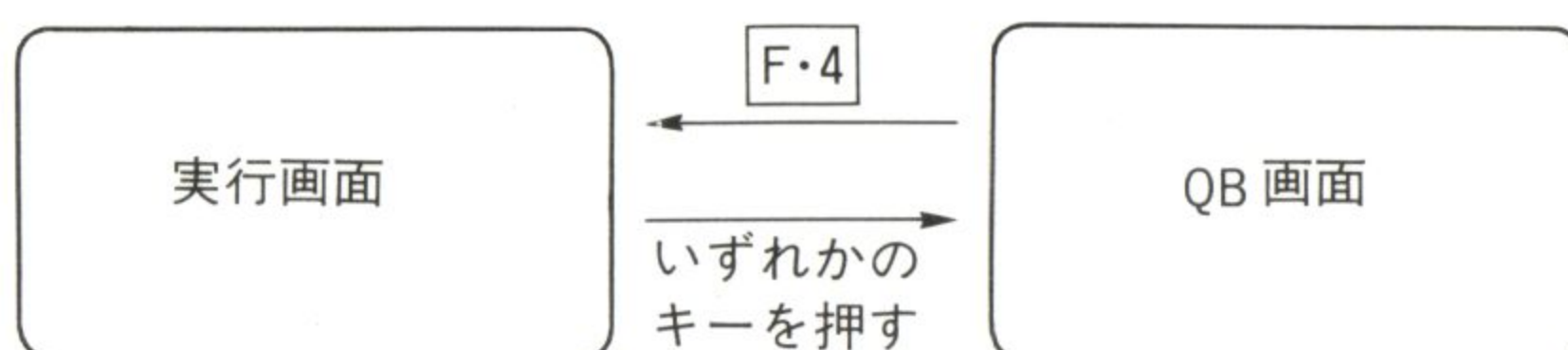
また，[CTRL]＋[F･10]によりアクティブ画面をフルスクリーンにしたり，元に戻したりすることができます．

2 アクティブ・ウィンドウの移動

F･6, SHIFT＋F･6によりアクティブ・ウィンドウを移動することができます.

マウスで移動する場合は，マウス・カーソルを移動したいウィンドウに移し，左ボタンをクリックします.

3 ウィンドウの拡大縮小

アクティブ・ウィンドウのサイズを拡大または縮小するには，GRPH＋＋，GRPH＋－を用います.

マウスを用いるならタイトル・バーにマウス・カーソルを合わせ，左ボタンを押しながらマウスを上/下に移動します. これで移動方向にウィンドウが拡大/縮小します.

4 実行画面への切り換え

プログラムの実行結果が出力されている画面とQB画面との切り換えは，F･4またはV/表示－U/実行画面で行います.

2−4 ファイル操作

1 プログラムのセーブ

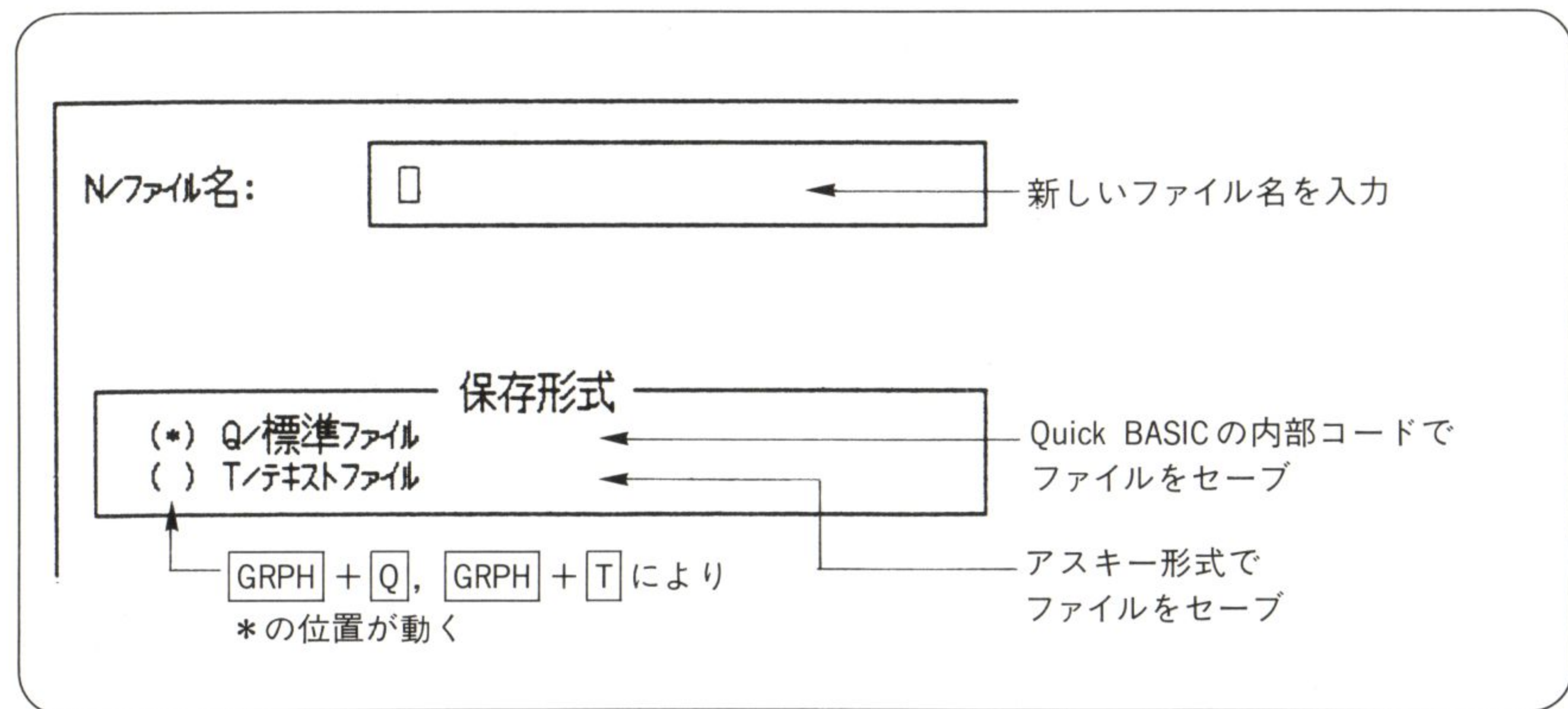
エディタ上で作ったファイルを保存するには，F/ファイルメニューの中の，次の3通りの方法があります．

S/保存 …現在のアクティブ・ウィンドウに表示されているファイルをセーブします．

V/すべて保存 …サブモジュールも含め，すべてをセーブします．マルチモジュールで作業しているときに便利です．

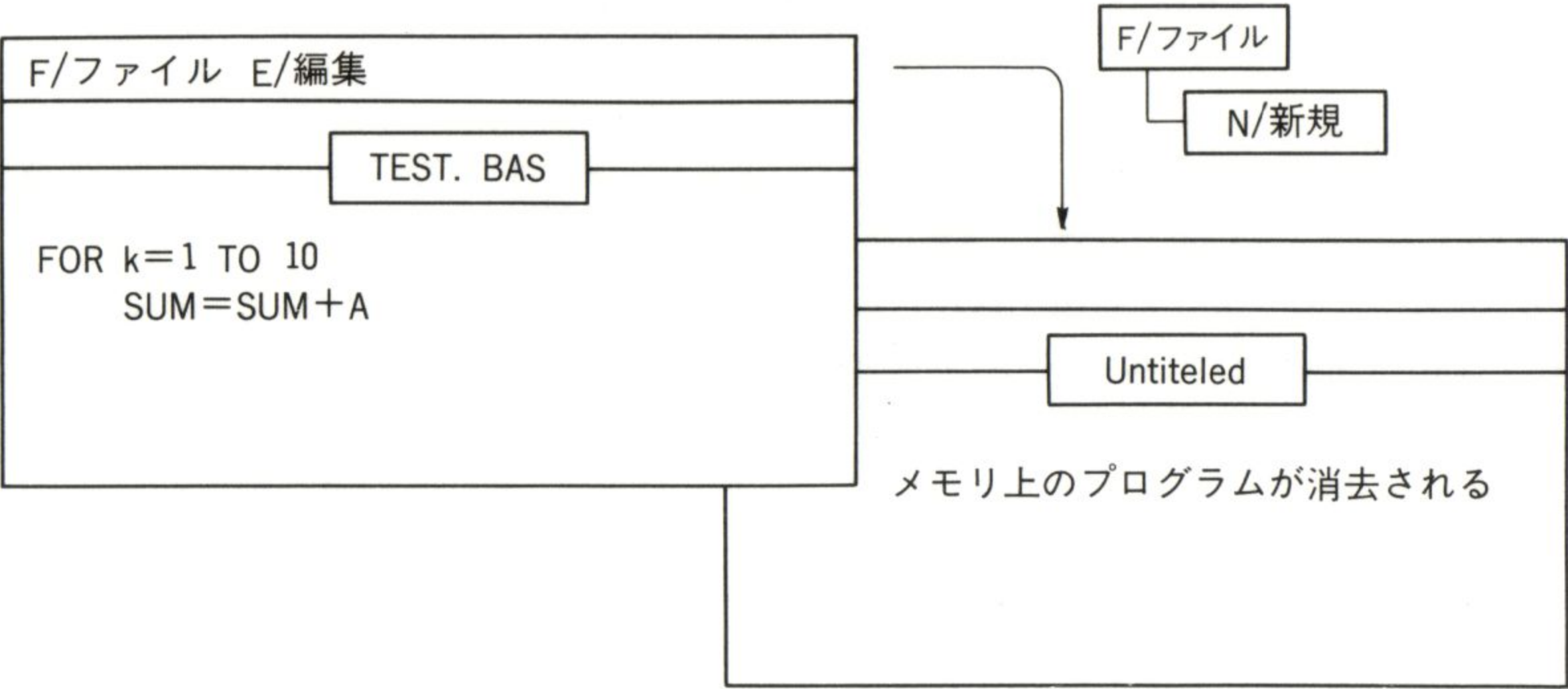
A/名前を変えて保存... …タイトル・バーに表示されている名前と異なる名前で保存します．

単一モジュールのプログラムの場合はS/保存，マルチモジュールのプログラムの場合はV/すべて保存を使うのが一般的です．

タイトル・バーが，「Untitled」(ファイル名がない場合)のときにS/保存かV/すべて保存を選択した場合，A/名前を変えて保存...を選択した場合は，次のようなダイアログ・ボックスが開くので，ファイル名およびファイルの保存形式を指定してください．

「Q/標準ファイル」の形式でセーブされたファイルはQuick BASICの内部コードでセーブされているので，TYPEコマンドで見ることはできませんし，ほかのエディタを用いて編集することもできません．

MS-DOS上のソーステキストとして一般性を持たせるには，「T/テキストファイル」の形式でセーブしてください．
一度セーブされたファイルをロードし再びセーブする場合は，前の保存形式がデフォルトとなります．

2 新規プログラムを作る

現在のプログラムの開発が終わり，次のプログラムを作りたいときは，一度Quick BASICを終了してから再起動してもよいのですが，QBの起動には時間がかかります．一般に[F/ファイル]メニューの[N/新規サブメニュー]を用いて，現在のプログラムをメモリ上から消し，次のプログラム開発に入ります．

3 プログラムのロード

エディタ上にプログラムをロードするには，F/ファイル メニューの中の次の2通りの方法があります．

O/読込...　…現在のエディタ上のプログラムを消してから指定されたプログラムをロードします．
M/追加...　…現在作業中のプログラムのカーソル位置の上部に挿入します．

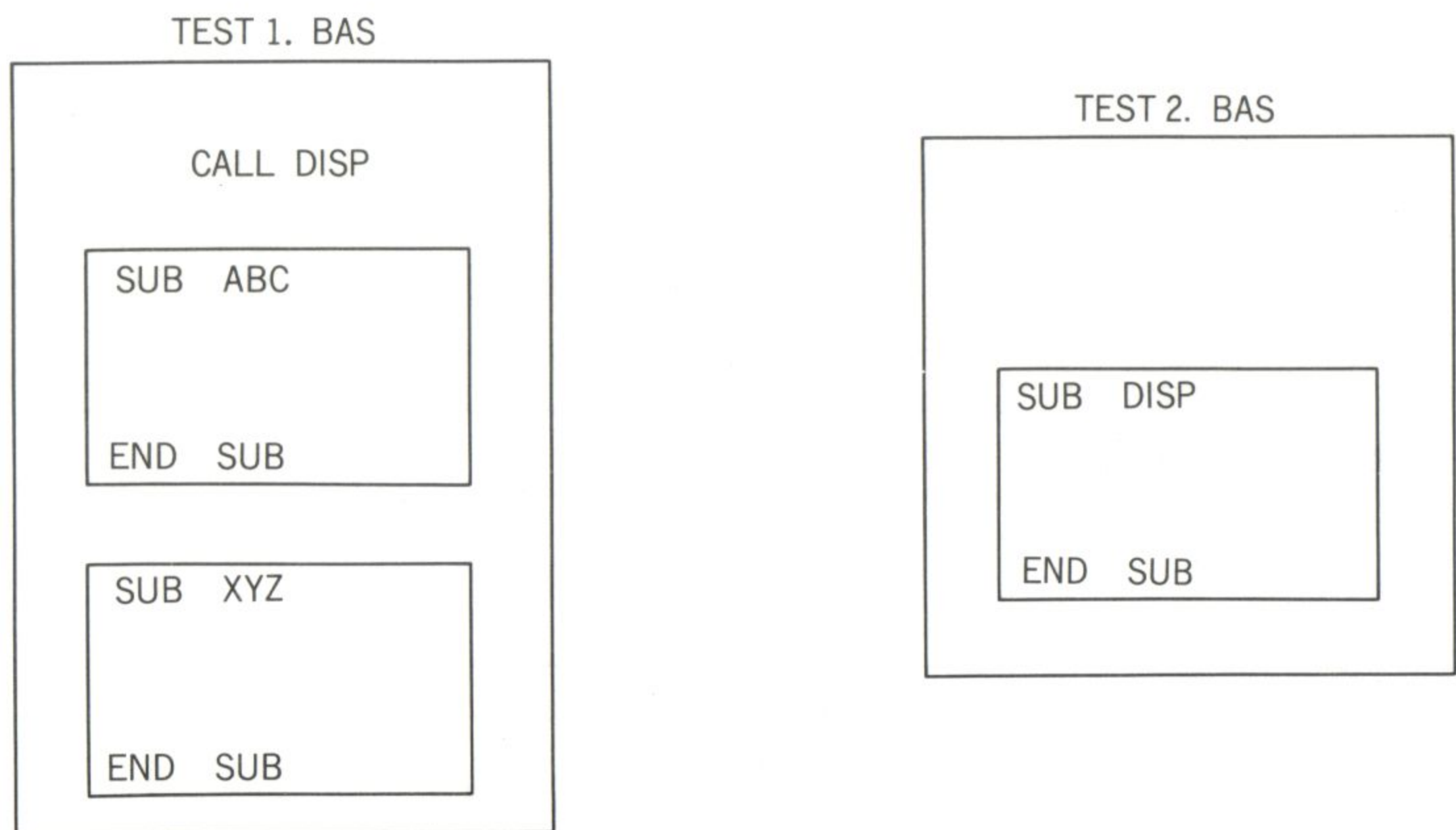
```
N/ファイル名:  *.BAS ←── ロードするファイル名

A:¥QB¥SOURCE

A.BAS ←──── カレントディレクトリにあるファイル一覧
[..]         ここにカーソルを移し ↵ で指定してもよい
[-A-]
[-B-]
[-C-]
[-D-]
```

4 プログラムのプリントアウト

F/ファイル ─ P/印刷... メニューにより，メモリ上のプログラムをプリンタに出力できます．次のようなダイアログ・ボックスが出るので，プリントアウトする範囲を指定してください．

```
─── 印刷 ───
( ) S/指定範囲            ←── ウィンドウ内で反転表示されているテキスト
( ) W/アクティブウインドウ ←── 表示されている画面
(*) M/カレントモジュール   ←── 作業中のモジュール
( ) A/全体                ←── メモリ上の全モジュール
```

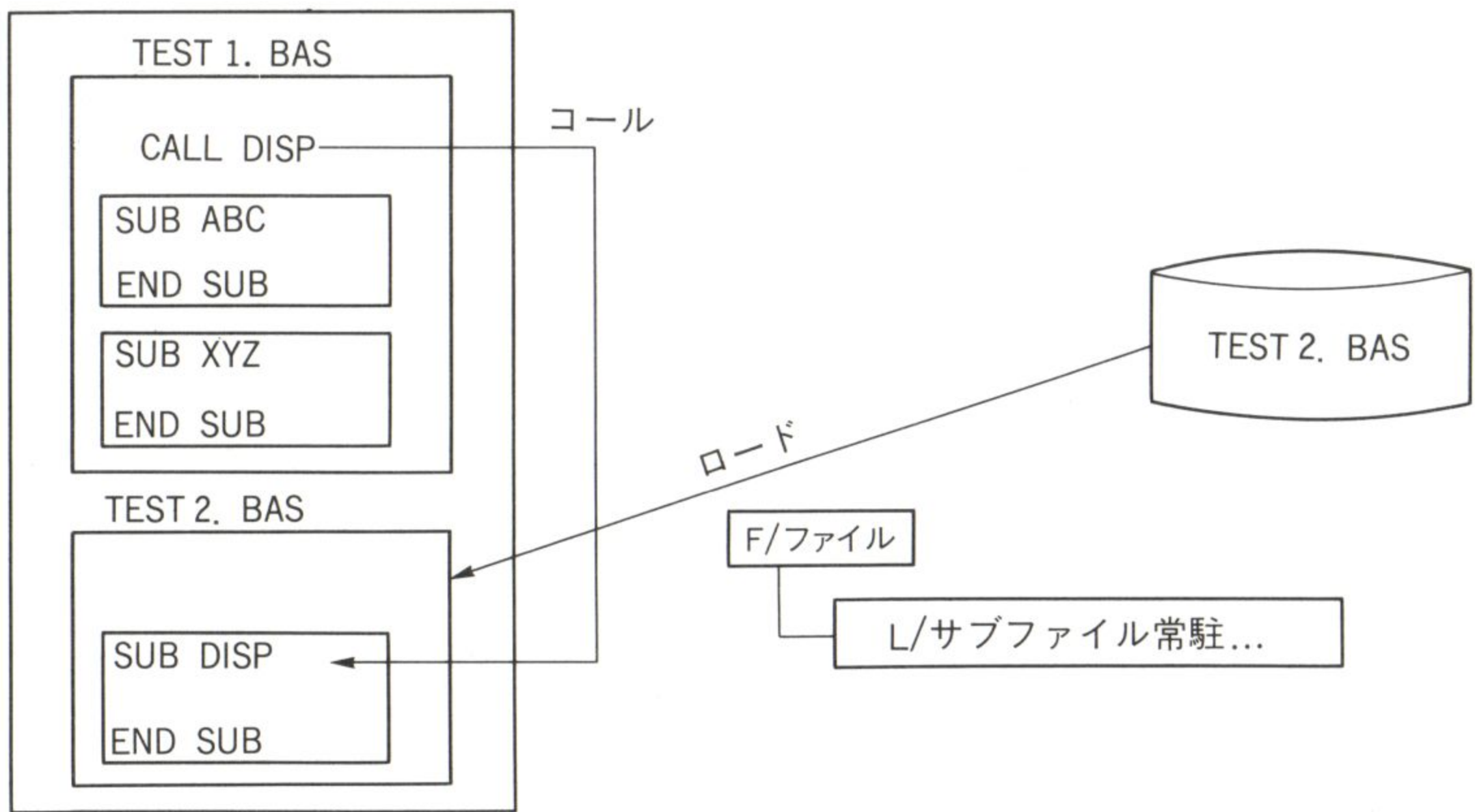
5 モジュール管理

たとえば，次のような２つのプログラムがあったとします．

QB では，複数のプログラムをメモリ上に並存させて管理することができます．メモリ上にロードされたプログラムをモジュールと呼びます．実行が開始されるモジュールをメインモジュールと呼び，他のモジュールをサブモジュールと呼びます．

QB 上にロードされているモジュールの各プロシージャ(SUB～END SUB, FUNCTION～END FUNCTION で定義されている)は，他のプログラム(モジュール)から利用することができます．

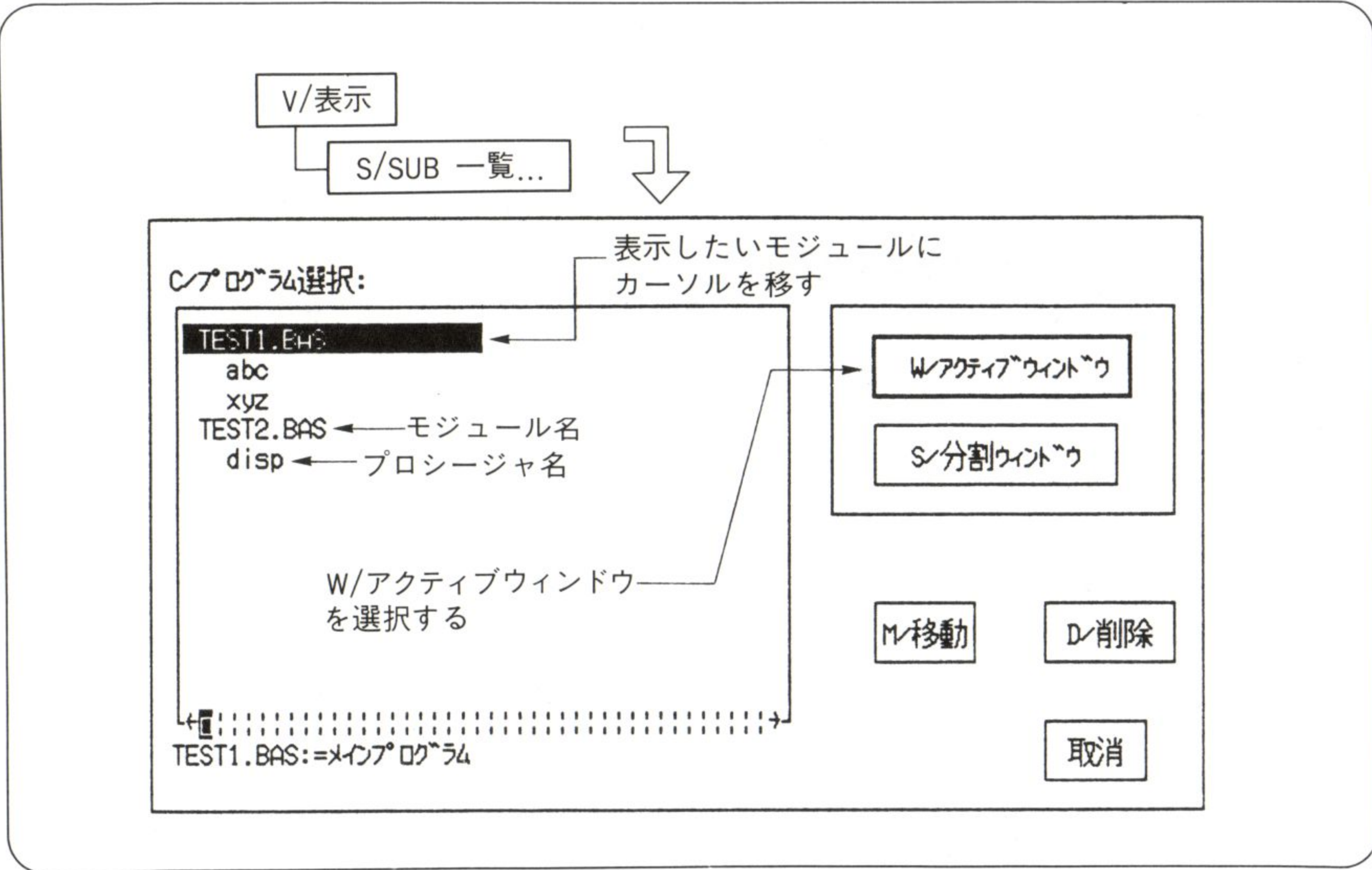
■サブファイルのロード

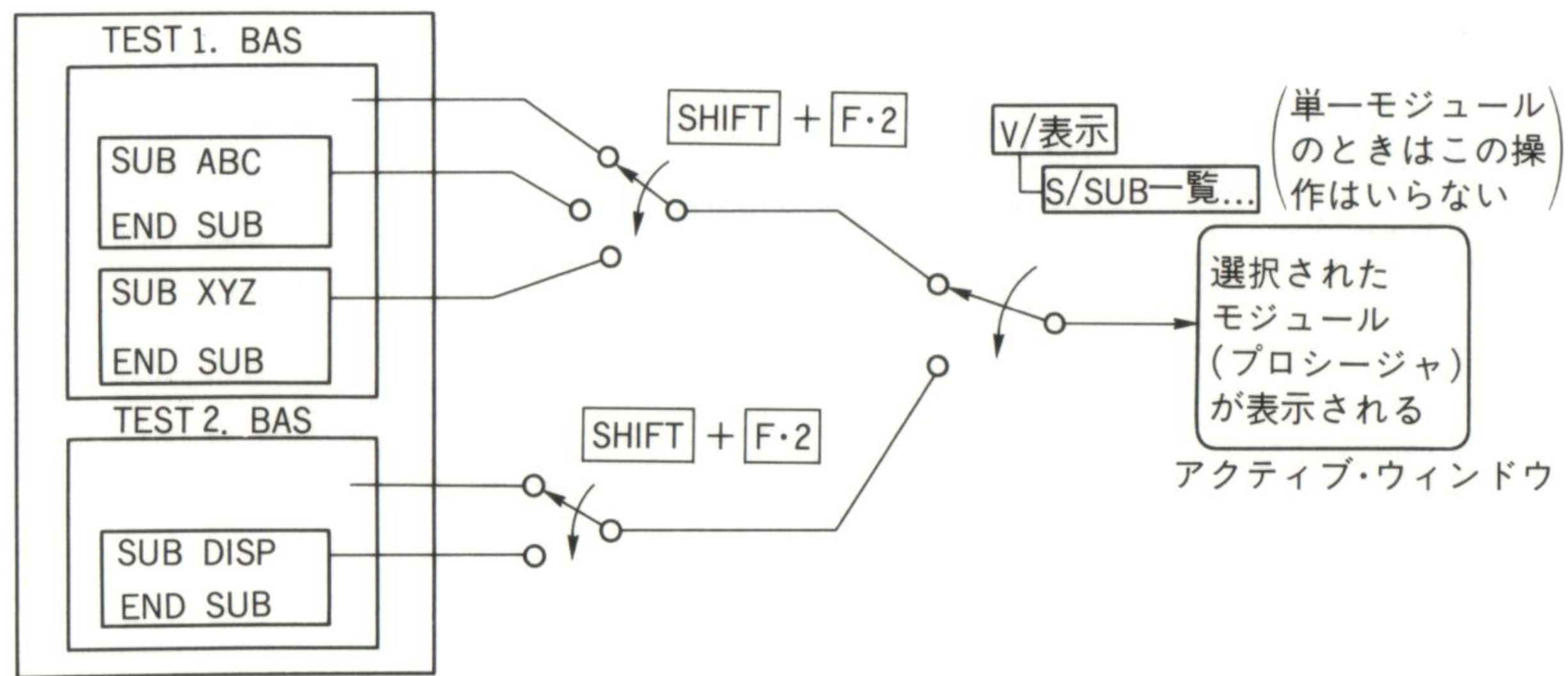
現在 TEST1.BAS を QB 上で編集しているものとし，TEST2.BAS という別のファイルを QB 上にロードするには，[F/ファイル]─[L/サブファイル常駐...]を選択します．

TEST 1. BAS
CALL DISP
SUB ABC
END SUB
SUB XYZ
END SUB
TEST 2. BAS
SUB DISP
END SUB
コール
ロード
TEST 2. BAS
F/ファイル
L/サブファイル常駐...

■モジュール画面の切り換え表示

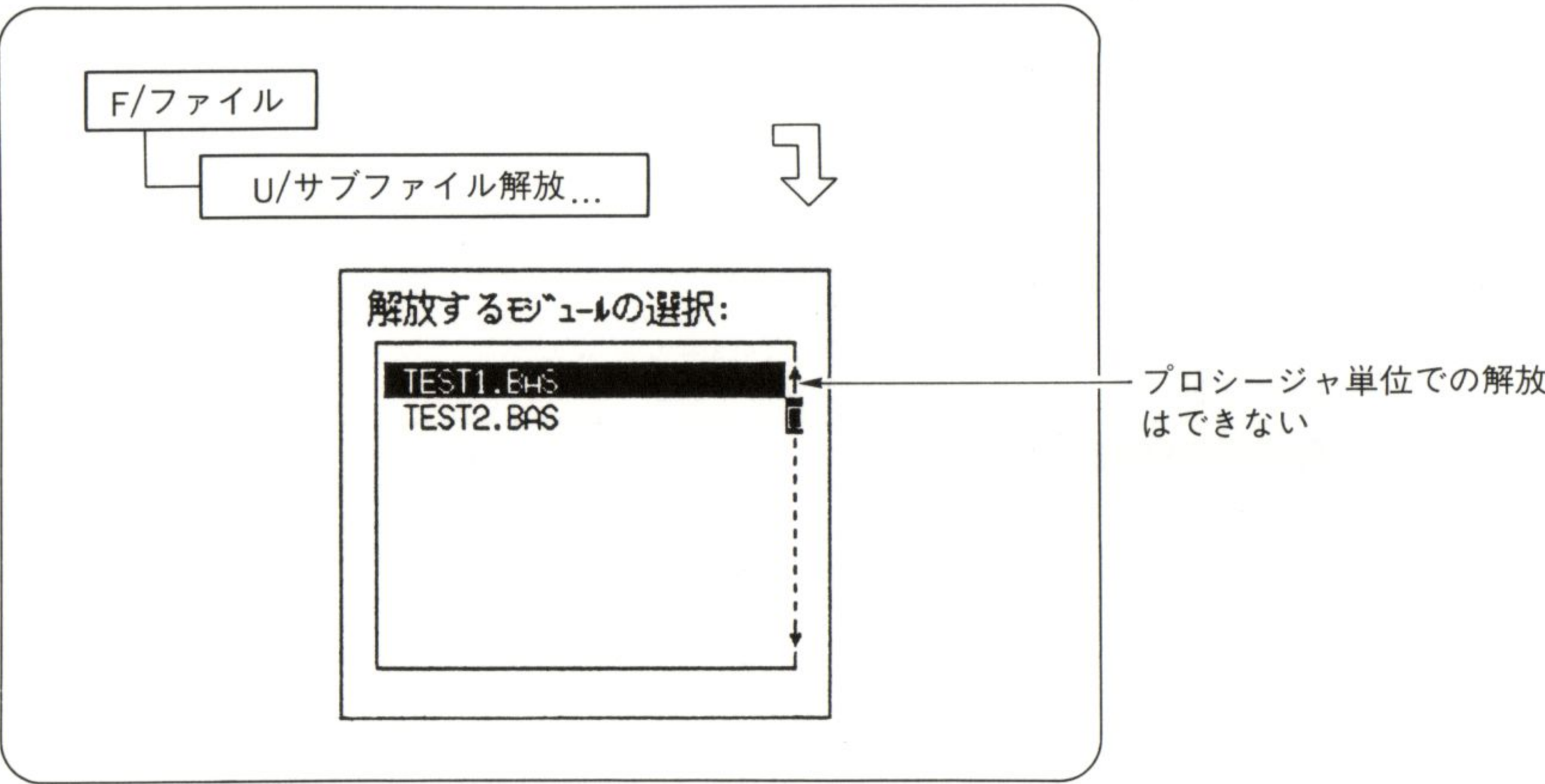
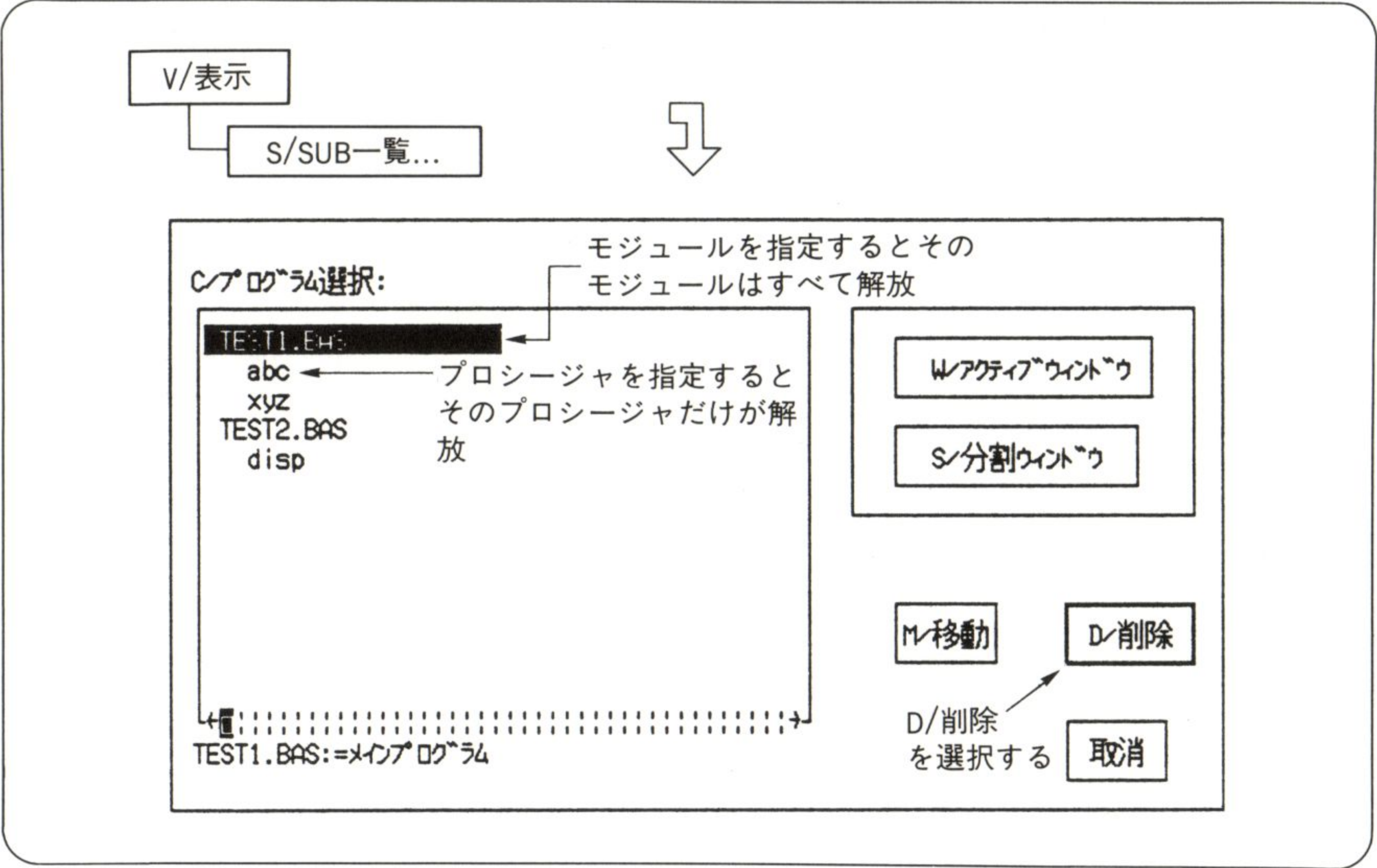
QB 上のモジュールおよびプロシージャはそれぞれ独立に管理されているため，同じ画面に同時に表示することはできません．

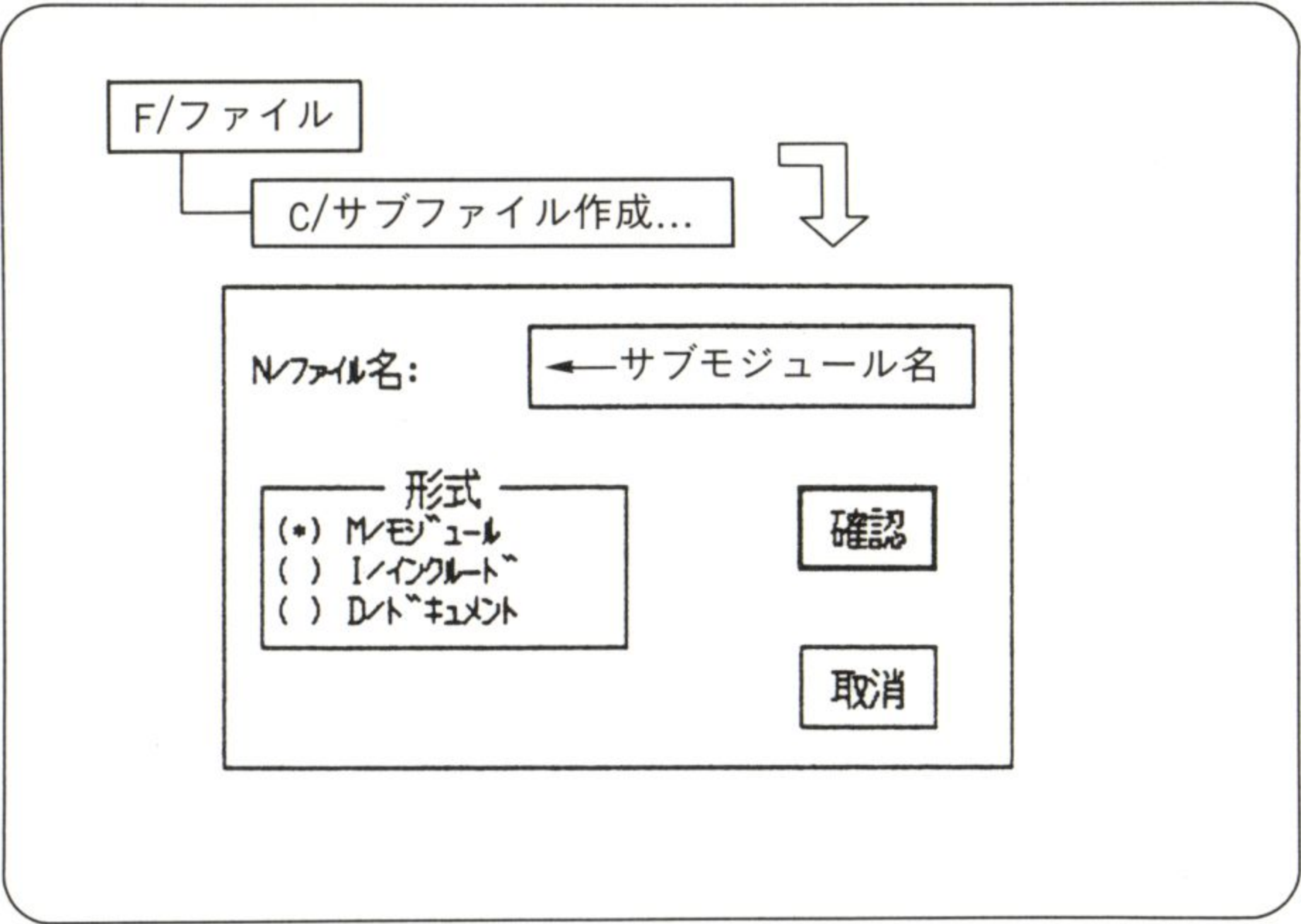
アクティブ・ウィンドウにどのモジュール(プロシージャ)を表示するかは，V/表示—S/SUB 一覧...または F・2 を用いて次のように行います．

V/表示
S/SUB 一覧...
C/プログラム選択:
TEST1.BAS
abc
xyz
TEST2.BAS ←モジュール名
disp ←プロシージャ名
表示したいモジュールにカーソルを移す
W/アクティブウィンドウ
S/分割ウィンドウ
W/アクティブウィンドウを選択する
M/移動
D/削除
取消
TEST1.BAS:=メインプログラム

なお，SHIFT＋F･2を用いれば，V/表示—S/SUB一覧...でアクティブになっているモジュール内の各プロシージャをアクティブ・ウィンドウに切り換え表示できます．

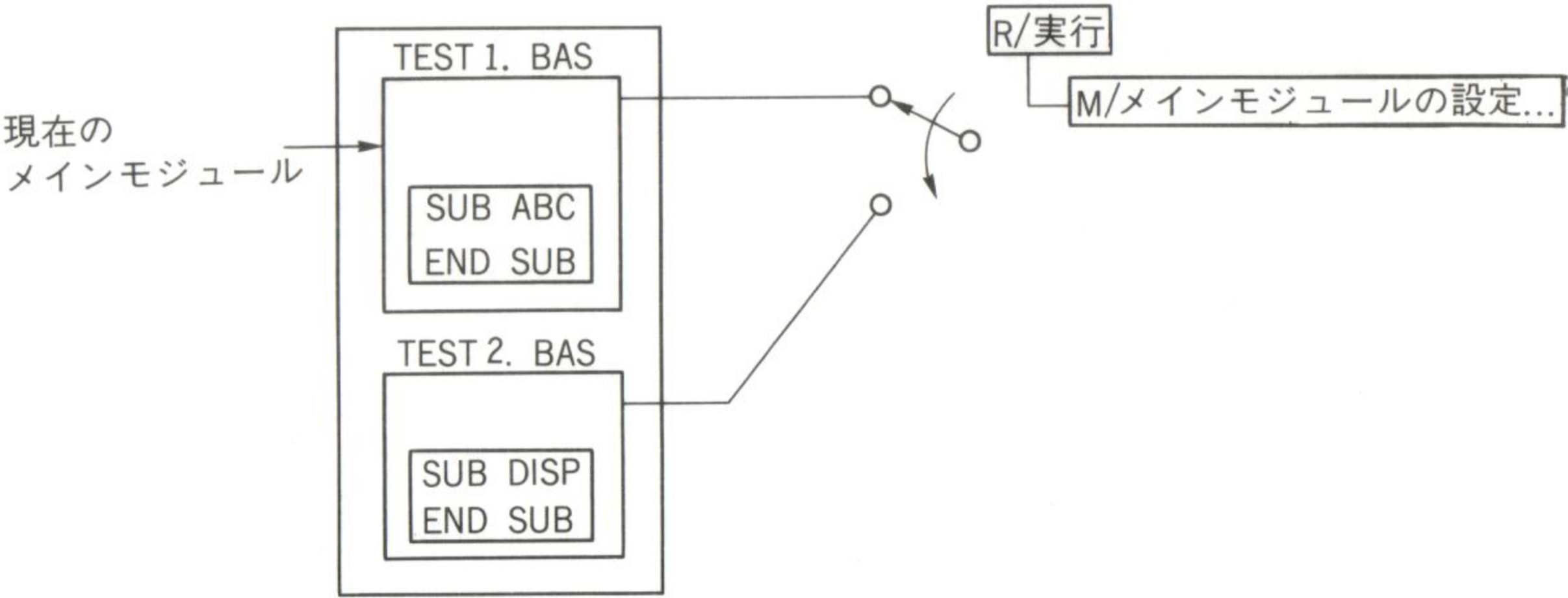
TEST 1. BAS
SUB ABC
END SUB
SUB XYZ
END SUB
TEST 2. BAS
SUB DISP
END SUB
SHIFT ＋ F･2
SHIFT ＋ F･2
V/表示
S/SUB一覧...
(単一モジュールのときはこの操作はいらない)
選択されたモジュール(プロシージャ)が表示される
アクティブ・ウィンドウ

■モジュールの解放

QB上にロードされているモジュールを削除する場合には，F/ファイル—U/サブファイル解放...または，V/表示—S/SUB一覧...のD/削除を用います．

F/ファイル
U/サブファイル解放...
解放するﾓｼﾞｭｰﾙの選択:
TEST1.BAS
TEST2.BAS
プロシージャ単位での解放はできない

■サブモジュールの作成

現在メモリ上にあるプログラムに新しくサブモジュールを加えるには，F/ファイル—C/サブファイル作成...を用います．

■メインモジュールの設定

QB上に複数のモジュールがロードされている場合，実行を開始するモジュール（メインモジュール）を[R/実行]—[M/メインモジュールの設定...]により切り換えることができます．

TEST 1. BAS
現在の
メインモジュール
SUB ABC
END SUB
TEST 2. BAS
SUB DISP
END SUB
R/実行
M/メインモジュールの設定...

6 ドキュメント・ファイルの作成

Quick BASICのエディタで，CONFIG.SYSやAUTOEXEC.BATなどのテキスト・ファイル(ドキュメント・ファイル)を作る場合，デフォルトモードでは構文チェック機能などが働くためうまくいきません．

テキスト・ファイルの作成法を以下に示します．

① F/ファイル — N/新規 でメモリ上のプログラムを消去する．

② F/ファイル — C/サブファイル作成... を選択し，次のダイアログ・ボックス内の各項目を設定する．

N/ファイル名：　CONFIG. SYS　← 作成するテキストの名前を入力

形式

( ) M/モジュール
( ) I/インクルード
(＊) D/ドキュメント　← GRPH ＋ D で ドキュメントを指定

③ テキストを入力する．

CONFIG. SYS

テキストを入力する

④ F/ファイル — S/保存 でテキストをセーブする．

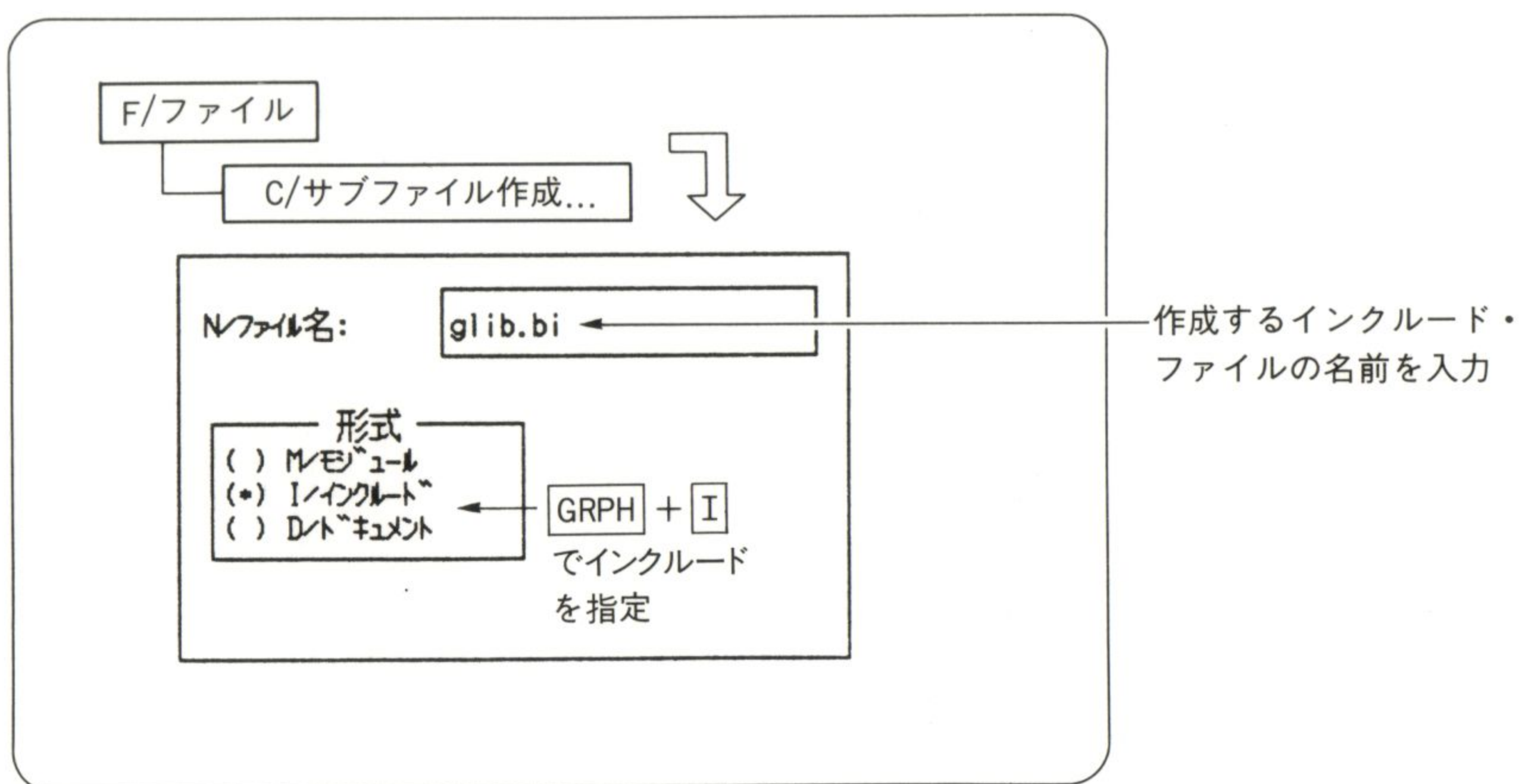
なお，すでに作成してあるテキスト・ファイル(ドキュメント・ファイル)を扱う場合は，上の②の代わりに F/ファイル — L/サブファイル常駐... を選択します．

7 インクルード・ファイル

インクルード・ファイルは，メタコマンドの「'$INCLUDE：'ファイル名'」で指定されるテキスト・ファイルです．

■インクルード・ファイルの作成

インクルード・ファイルを作成するのは，ドキュメント・ファイルの作成方法と同様ですが，F/ファイル—C/サブファイル作成...を選択し，「I/インクルード」形式を指定します．

F/ファイル
C/サブファイル作成...
N/ファイル名:
glib.bi
作成するインクルード・ファイルの名前を入力
形式
( ) M/モジュール
(*) I/インクルード
( ) D/ドキュメント
GRPH + I
でインクルードを指定

■インクルード・ファイルの表示

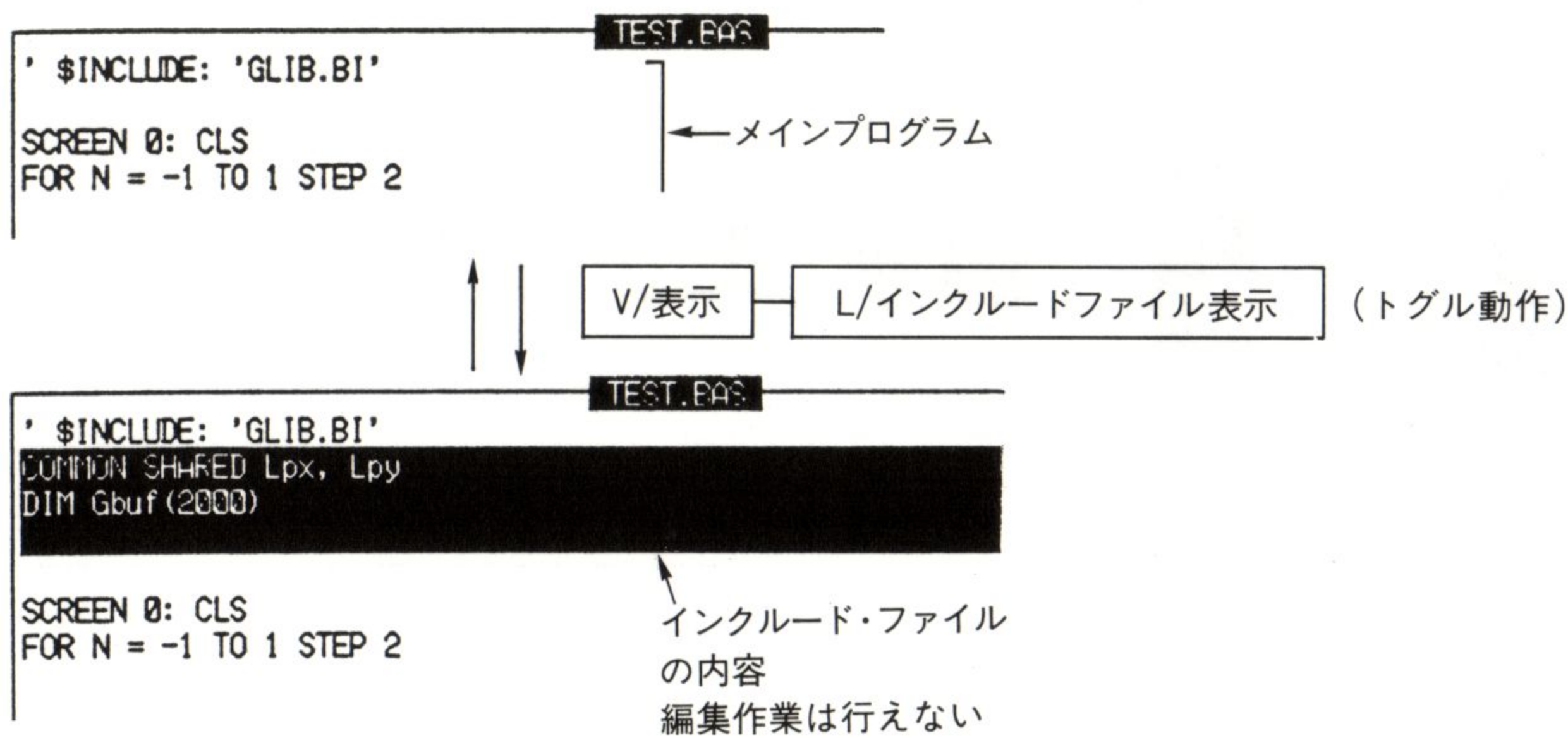
メインプログラム中の$INCLUDEで指定されているインクルード・ファイルの内容を表示するには，V/表示—L/インクルードファイル表示を選択します．逆に表示を消す場合にもこのメニューを選択します．

```
TEST.BAS
' $INCLUDE: 'GLIB.BI'

SCREEN 0: CLS
FOR N = -1 TO 1 STEP 2
```

←メインプログラム

V/表示 ─ L/インクルードファイル表示 （トグル動作）

```
TEST.BAS
' $INCLUDE: 'GLIB.BI'
COMMON SHARED Lpx, Lpy
DIM Gbuf(2000)

SCREEN 0: CLS
FOR N = -1 TO 1 STEP 2
```

インクルード・ファイルの内容
編集作業は行えない

■インクルード・ファイルの編集

カーソルを$INCLUDEメタコマンドのある行に置いて，V/表示 ─ I/インクルードファイル編集を選択します．

カーソル

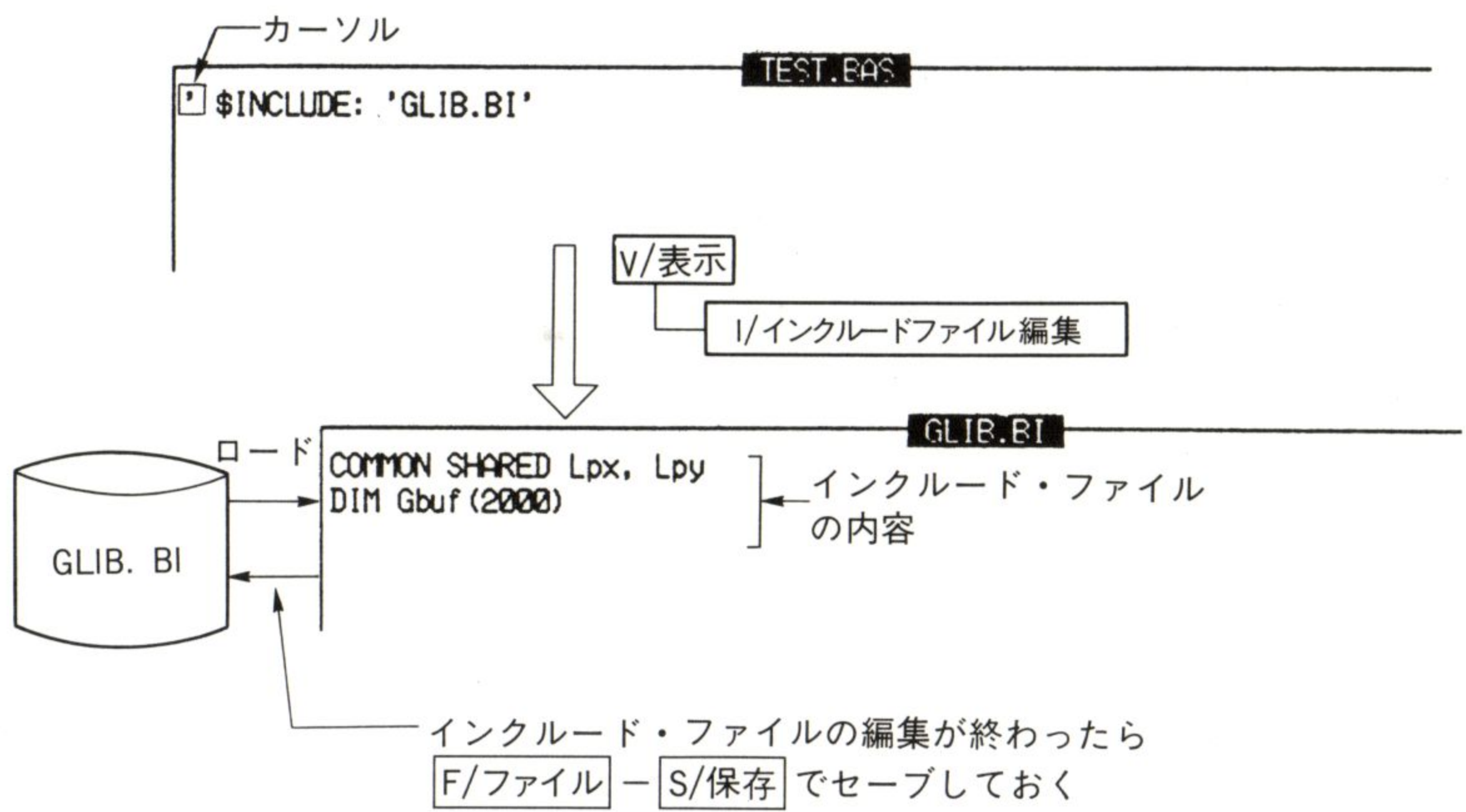
```
TEST.BAS
' $INCLUDE: 'GLIB.BI'
```

V/表示 ─ I/インクルードファイル編集

GLIB. BI　ロード

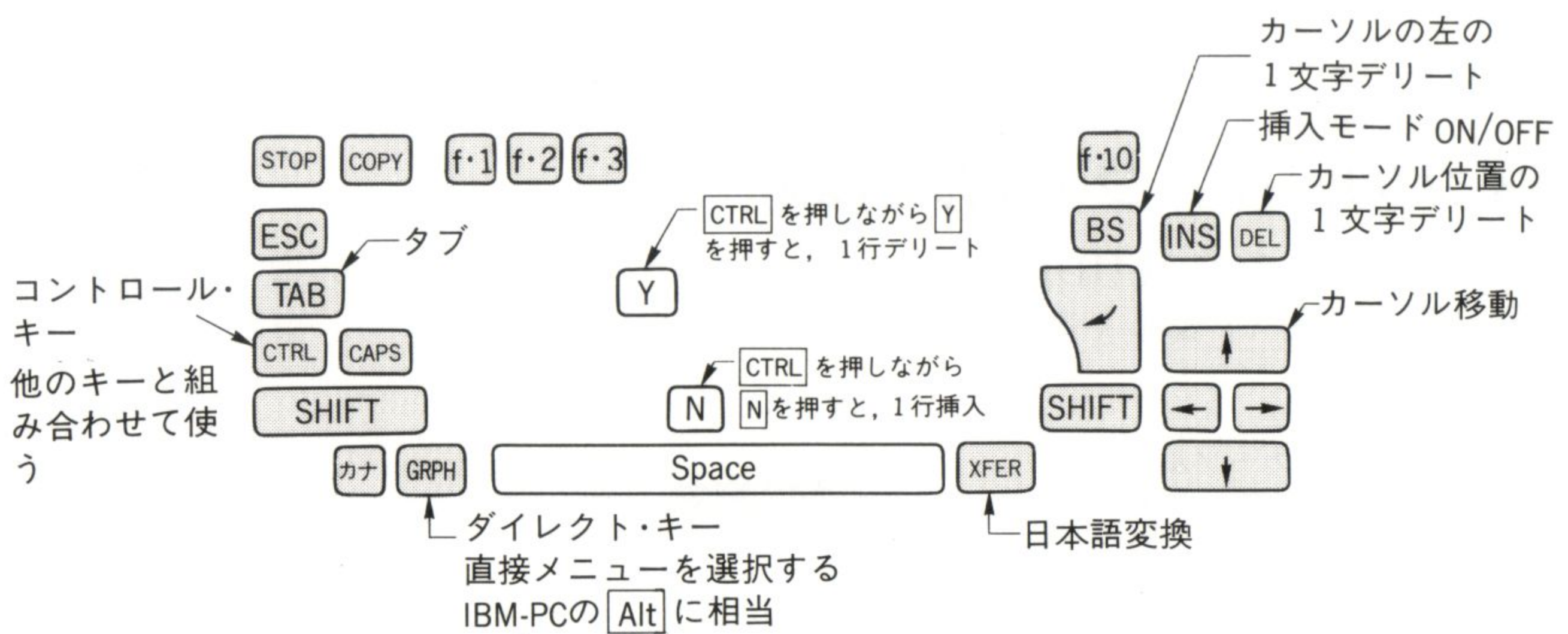
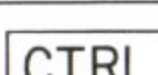
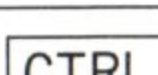
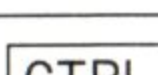
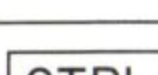
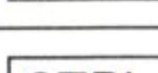
```
GLIB.BI
COMMON SHARED Lpx, Lpy
DIM Gbuf(2000)
```

インクルード・ファイルの内容

インクルード・ファイルの編集が終わったら
F/ファイル ─ S/保存 でセーブしておく

なお，すでにインクルード・ファイルがメモリに在駐している場合は，V/表示 ─ S/SUB一覧...によりインクルード・ファイルをアクティブウィンドウに表示して，編集作業に入ってください．

2−5 編集(エディタ)

1 キー配置

エディタ上で使う重要なキーの配置を以下に示します.

2 編集コマンド一覧

テキスト編集で使うキー・コマンドの一覧を以下に示します.

表2-4　キー・コマンド一覧

	編集キー	機　能
カーソルの移動	←	カーソルを左に1文字移動
	→	カーソルを右に1文字移動
	↑	カーソルを上に1行移動
	↓	カーソルを下に1行移動
	CTRL＋←	左に1単語分移動
	CTRL＋→	右に1単語分移動
	HOME/CLR	カレント行の最初の文字に移動
	CTRL＋Q S	カレント行の物理的な左端に移動
	CTRL＋↵	次の行の最初の文字に移動
	HELP	カレント行の最後の文字に移動
	CTRL＋Q E	ウィンドウの上端に移動
	CTRL＋Q X	ウィンドウの下端に移動
	CTRL＋Q R	モジュール/プロシージャの先頭に移動
	CTRL＋Q C	モジュール/プロシージャの終わりに移動
	CTRL＋K n	カレント行にn番(0〜3の数)のマークを設定
	CTRL＋Q n	n番(0〜3の数)のマーク位置に移動
挿入	INS	挿入モードのON/OFF
	CTRL＋N	カレント行の上に1行挿入.カーソルが行の先頭位置にないとその行が中断されるので,HOME/CLRでカーソルを先頭文字に移してからCTRL＋Nを行う
	↵	カレント行の下に1行挿入.カーソルが行の終末位置にないとその行が中断されるので,その場合はHELPでカーソルを行の終末文字に移してから↵を行う

	編集キー	機　　能
削除	CTRL + Y	カーソル行を削除(この内容はクリップボードに保存される)
	CTRL + Q Y	カーソル位置から行末までを削除(この内容はクリップボードに保存される)
	BS	カーソル位置の左の文字を削除
	DEL	カーソル位置の文字を削除
	CTRL + T	1単語を削除
クリップボード	SHIFT + INS	クリップボードの内容をカーソル位置にコピー
	SHIFT + DEL	指定した(反転表示された)テキストを削除し,クリップボードにセーブ
	CTRL + INS	指定した(反転表示された)テキストを削除せず,そのままにしてクリップボードにセーブ
タブ	TAB	次のタブ位置に移動
	SHIFT + BS	バック・タブ(1つ前のタブ位置に戻る)
スクロール	CTRL + W CTRL + Z ROLL DOWN ROLL UP CTRL + ROLL UP CTRL + ROLL DOWN	上に1行分スクロール 下に1行分スクロール 上に1画面分スクロール 下に1画面分スクロール 左に1画面分スクロール 右に1画面分スクロール
テキストの指定	SHIFT ＋カーソルの移動コマンド・キー	SHIFT を押しながらカーソルの移動コマンド・キー(← ～ CTRL + Q C)を押すと,カーソルが移動した範囲が反転表示され,テキストが指定されたことになる
サーチ	F・3	次に一致する検索文字列を探す
	CTRL + ¥	指定したテキストを探す
	CTRL + Q A	指定したテキストを探し,新しいテキストで置き換える

3 自動構文チェック機能

Quick BASICのエディタは，自動構文チェック機能が付いています．[↵]または[↑]，[↓]を押すと，その行の内容が正しいBASIC文法にしたがっているかチェックし，もし誤っていればメッセージを出してくれます．

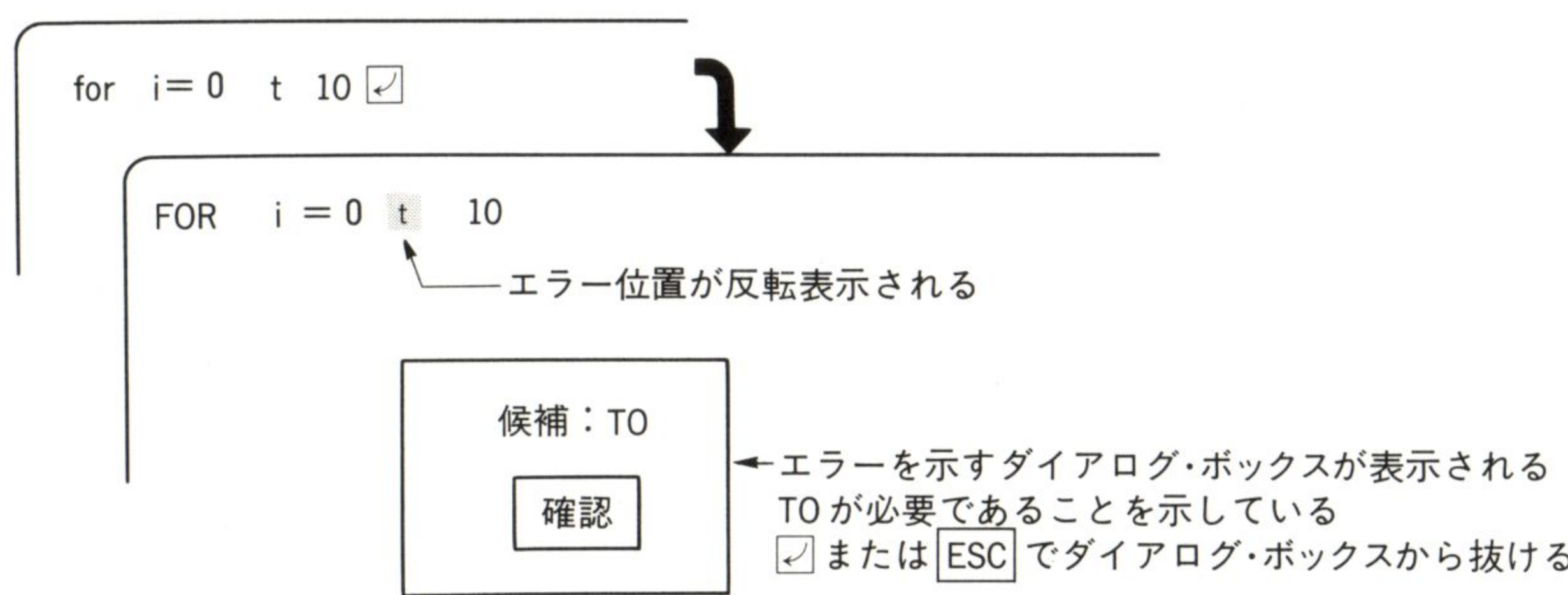
```
for  i=0  t  10↵
```

```
FOR  i = 0  t  10
```

エラー位置が反転表示される

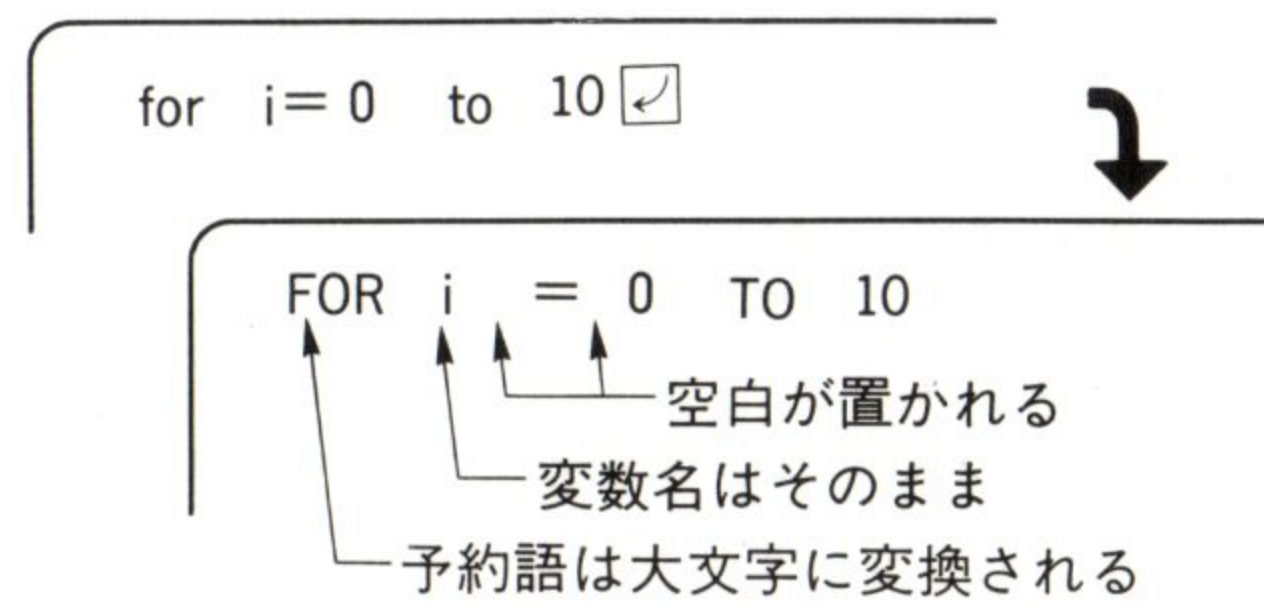
候補：TO

確認

エラーを示すダイアログ・ボックスが表示される
TOが必要であることを示している
[↵]または[ESC]でダイアログ・ボックスから抜ける

1行の内容が正しければ，予約語(for，ifなど)を大文字に変換し，演算子の両脇に空白を補います．また，インタプリタがその行の内容をBASICの内部コードに変換して，メモリに格納します．

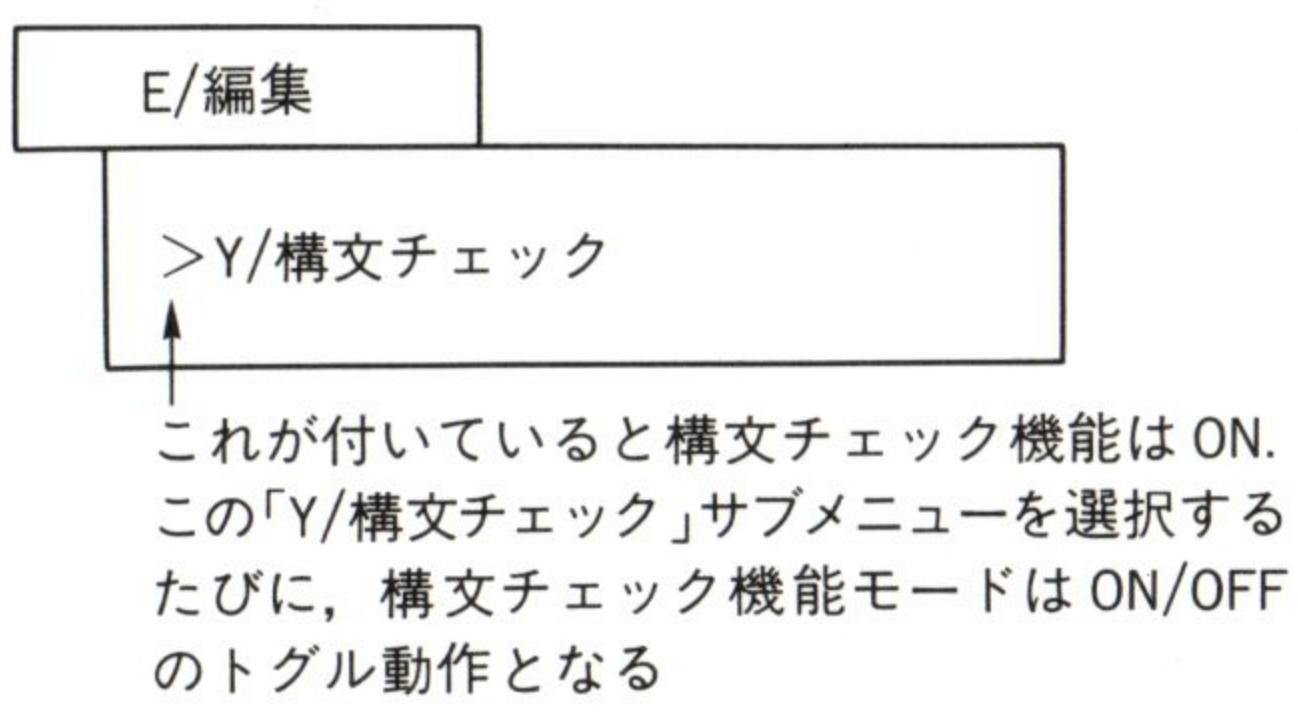
```
for  i=0  to  10↵
```

```
FOR i  =  0  TO  10
```

空白が置かれる
変数名はそのまま
予約語は大文字に変換される

Quick BASICのエディタには自動構文チェックが付いているため，その行に誤りがあればエラーメッセージが表示され，入力作業を中断してしまいます．もし，このようなていねいなサポートがきらいなら，自動構文チェック機能をOFFにしておけばよいでしょう．

E/編集

>Y/構文チェック

これが付いていると構文チェック機能はON.
この「Y/構文チェック」サブメニューを選択するたびに，構文チェック機能モードはON/OFFのトグル動作となる

注）Ver. 4.5では，[O/オプション]の[Y/構文チェック]を指定します．
　また>ではなく＊という表示になりました．

4 オート・インデント機能

プログラムを見やすくするために空白やタブを適当に入れ，while や for などのループ構造や，if then else などのブロックがひと目でわかるように，書き出す先頭位置を右に下げます．これをインデント(字下げ)といいます．

インデントの間隔は何文字でもかまいませんが，通常は TAB の間隔でとっていきます．Quick BASIC の TAB 間隔は 4 文字です(V/表示 メニューの O/オプション サブメニューで変更できます)．

注)Ver.4.5では，O/オプション の D/画面設定 を指定します。

```
FOR    i=1    TO    10
       SUM=0
       FOR    j=1    TO    N
              SUM=SUM+A
 TAB     TAB
 ───→   ───→          ⋮
 4文字   4文字
       NEXT j
NEXT i
```

オート・インデント機能とは，次のように ↵ で次の行に移ったときに，直前の行の先頭文字位置に自動的に頭合わせをすることをいます．

```
FOR    i=1    TO    10↵
   TAB    FOR    j=1    TO    N↵
          █
          ↑
          └──── 直前の行の先頭文字位置にカーソルがくる(オート・インデント)
```

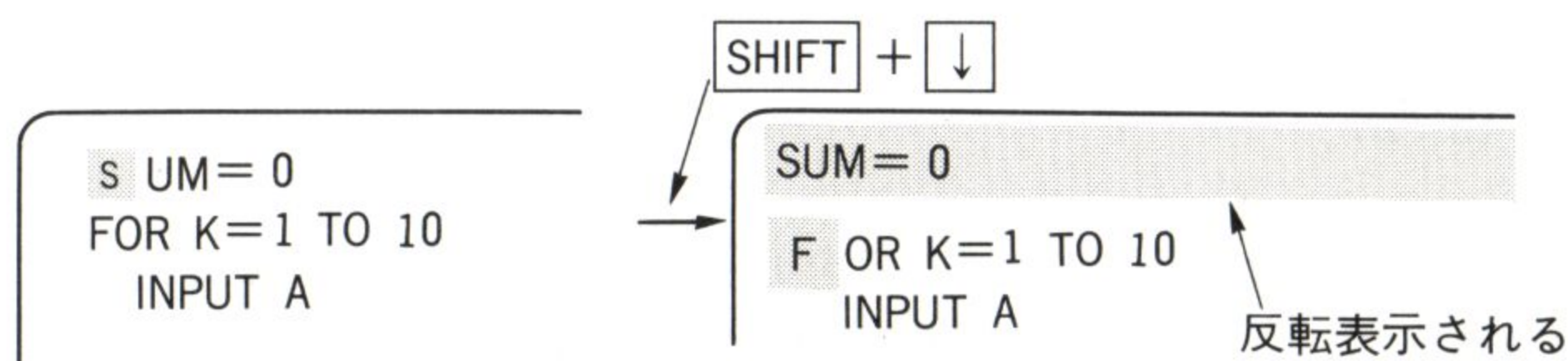
5 テキストの指定

クリップボードを介したテキストのムーブ/コピーでは，対象となるブロック(テキスト)を指定しなければなりません．SHIFT＋カーソルの移動コマンド・キー(←～CTRL＋Q C)を押すと，カーソルが移動した範囲が反転表示され，テキストが指定されたことになります．

テキストが指定されている状態でE/編集―E/削除またはDELを行うと，そのテキスト全体が削除されます．

```
SUM＝0
FOR K＝1 TO 10
  INPUT A
```

SHIFT＋↓

```
SUM＝0
FOR K＝1 TO 10
  INPUT A
```

反転表示される

マウスを用いてテキストを指定するには，マウスの左ボタンを押しながらマウスを移動します．マウス・カーソルが移動した範囲が反転表示され，テキストを指定したことになります．

6 テキストのブロック・コピー/ムーブ

クリップボードという Quick BASIC が確保している作業領域を介して，テキストのブロック・コピー，ブロック・ムーブを行うことができます．テキストのブロック・コピーは，CTRL＋INSとSHIFT＋INSを用いて次のように行います．

A＝0

反転表示されたブロック

```
SUM＝SUM＋DAT
   ⋮
PRINT SUM
```

FOR K＝1 TO 10

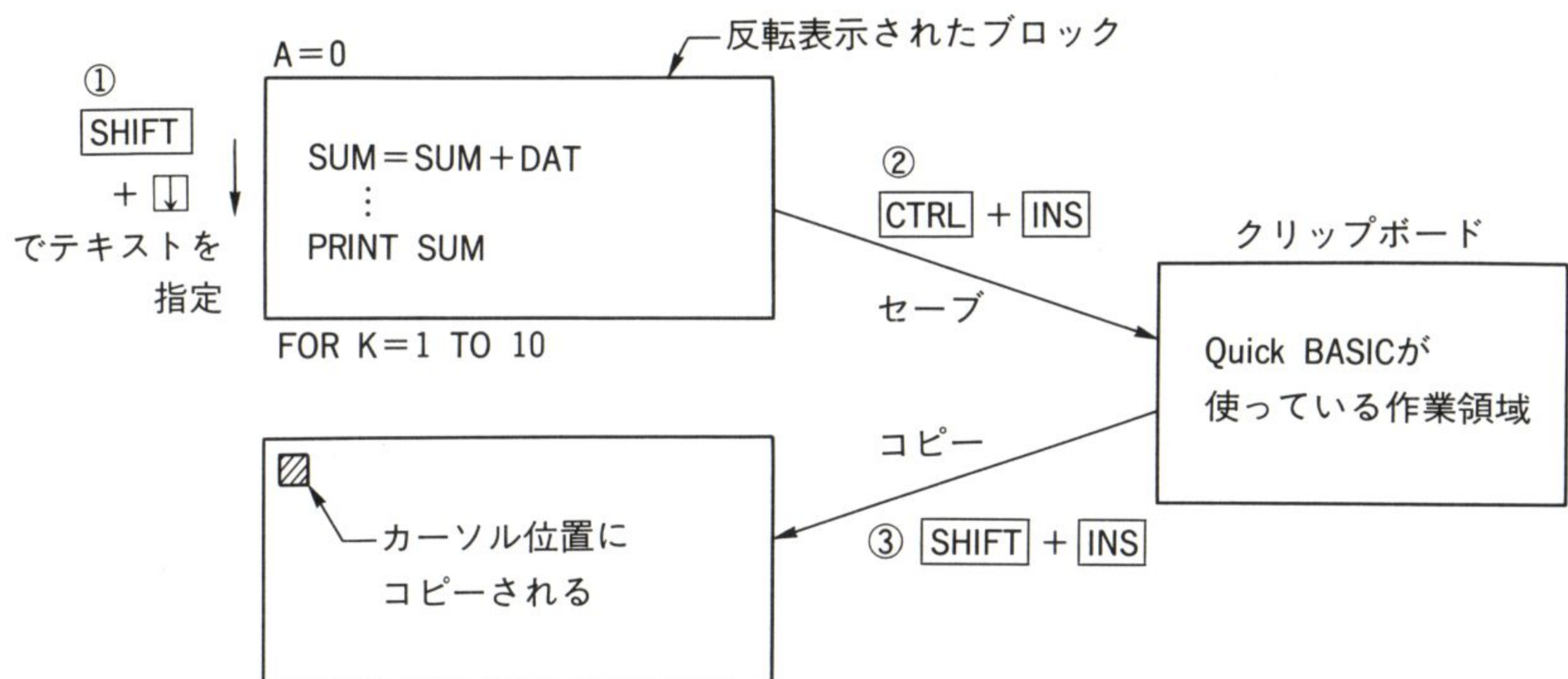
① SHIFT ＋↓ でテキストを指定

② CTRL＋INS セーブ

クリップボード

Quick BASICが使っている作業領域

コピー ③ SHIFT＋INS

カーソル位置にコピーされる

これに対してブロック・ムーブは指定テキストを削除して，別の位置に移動(ムーブ)するので，SHIFT＋DELとSHIFT＋INSを用いて行います．

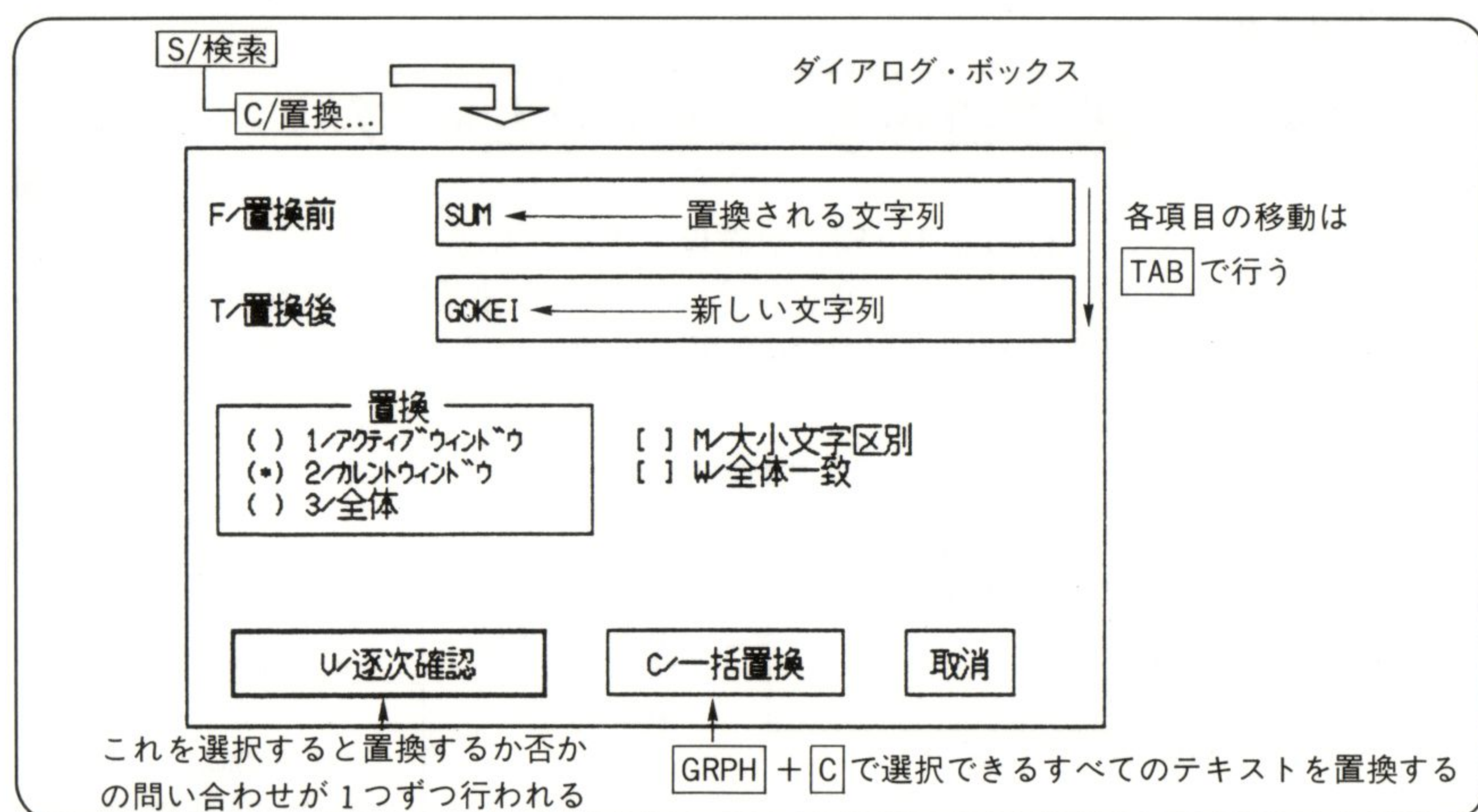
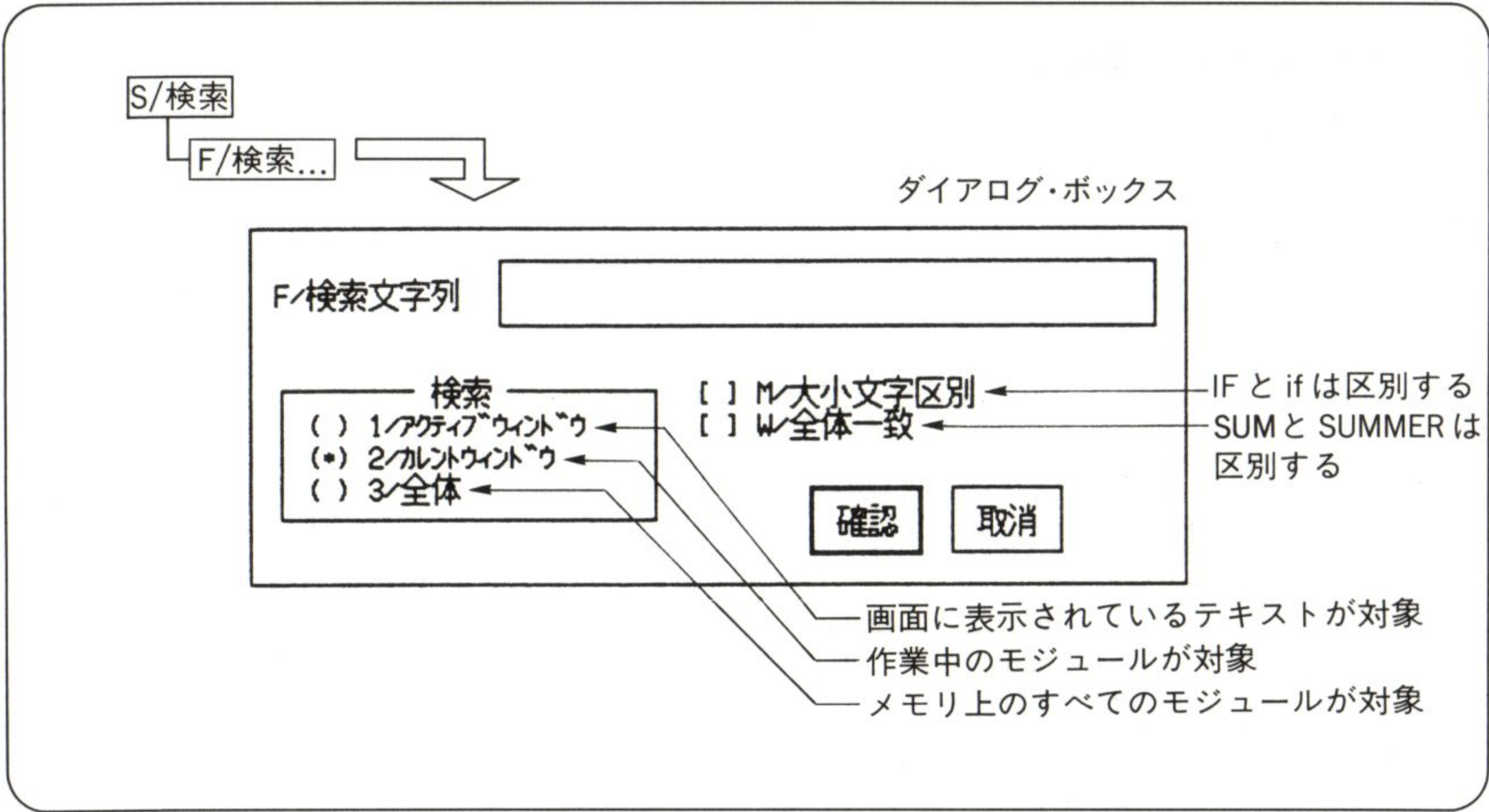
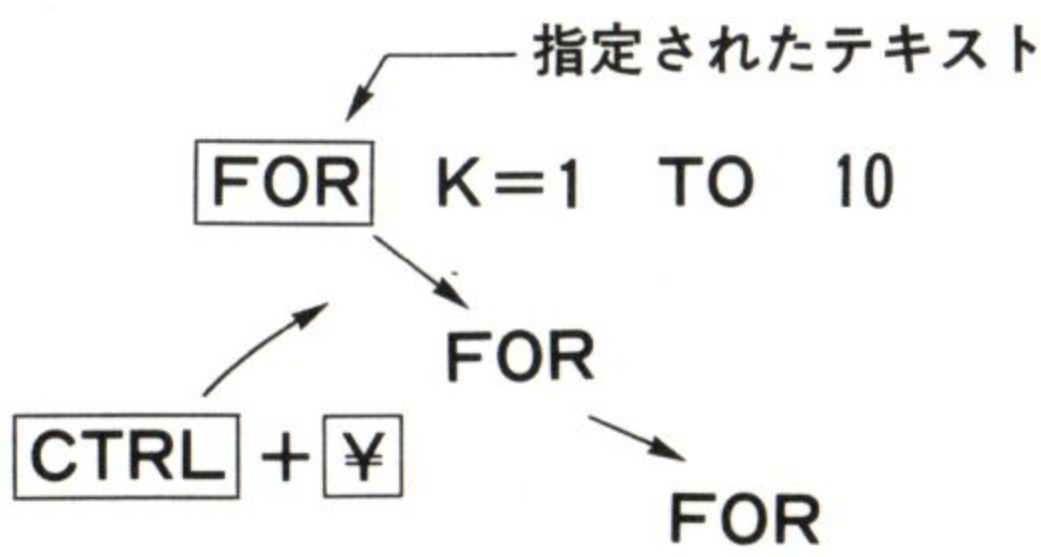
7 文字列のサーチとリプレイス

たとえば，次のようなテキストがあったとします．

```
SUM=0
WHILE   SUM<100
        INPUT  A
        SUM=SUM+A
WEND
PRINT   SUM
```

なんらかの都合で，SUMという文字をGOKEIに変える必要が生じたとき，いちいちSUMを探しだしてGOKEIに打ち直していてはたいへんですし，見落としも生じるでしょう．このようなときは，文字列のリプレイス・コマンド(S/検索—C/置換...)を用いて次のように行います．

この操作でテキスト中のSUMは一瞬にしてGOKEIに置き換えられます．

S/検索
C/置換...
ダイアログ・ボックス
F/置換前　SUM ← 置換される文字列
T/置換後　GOKEI ← 新しい文字列
各項目の移動はTABで行う
置換
( ) 1/アクティブウィンドウ
(*) 2/カレントウィンドウ
( ) 3/全体
[ ] M/大小文字区別
[ ] W/全体一致
U/逐次確認　C/一括置換　取消
これを選択すると置換するか否かの問い合わせが1つずつ行われる
GRPH+Cで選択できるすべてのテキストを置換する

文字列のサーチは，S/検索—F/検索...を用いて次のように行います．次の文字列をサーチするにはF･3を用います．

SHIFT＋矢印キーにより指定されたテキスト(反転表示されている)をサーチするには，S/検索—S/指定文字列検索またはCTRL＋¥を用います.

8 ラベルの検索

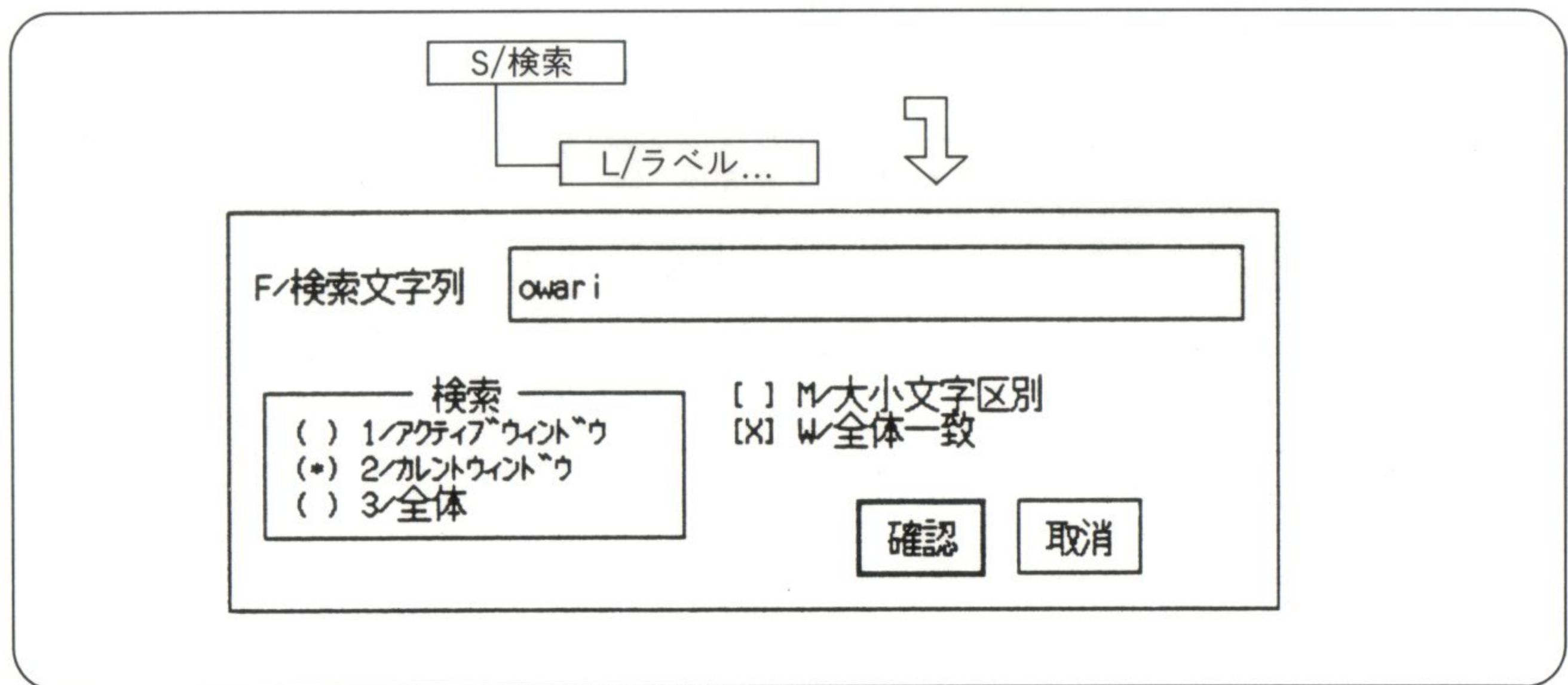
ラベルの検索は[S/検索]—[L/ラベル...]を用います.

S/検索
L/ラベル...

F/検索文字列 owari

検索
( ) 1/アクティブウィンドウ
(*) 2/カレントウィンドウ
( ) 3/全体

[ ] M/大小文字区別
[X] W/全体一致

確認　取消

9 その他の機能

■プレース・マーカ

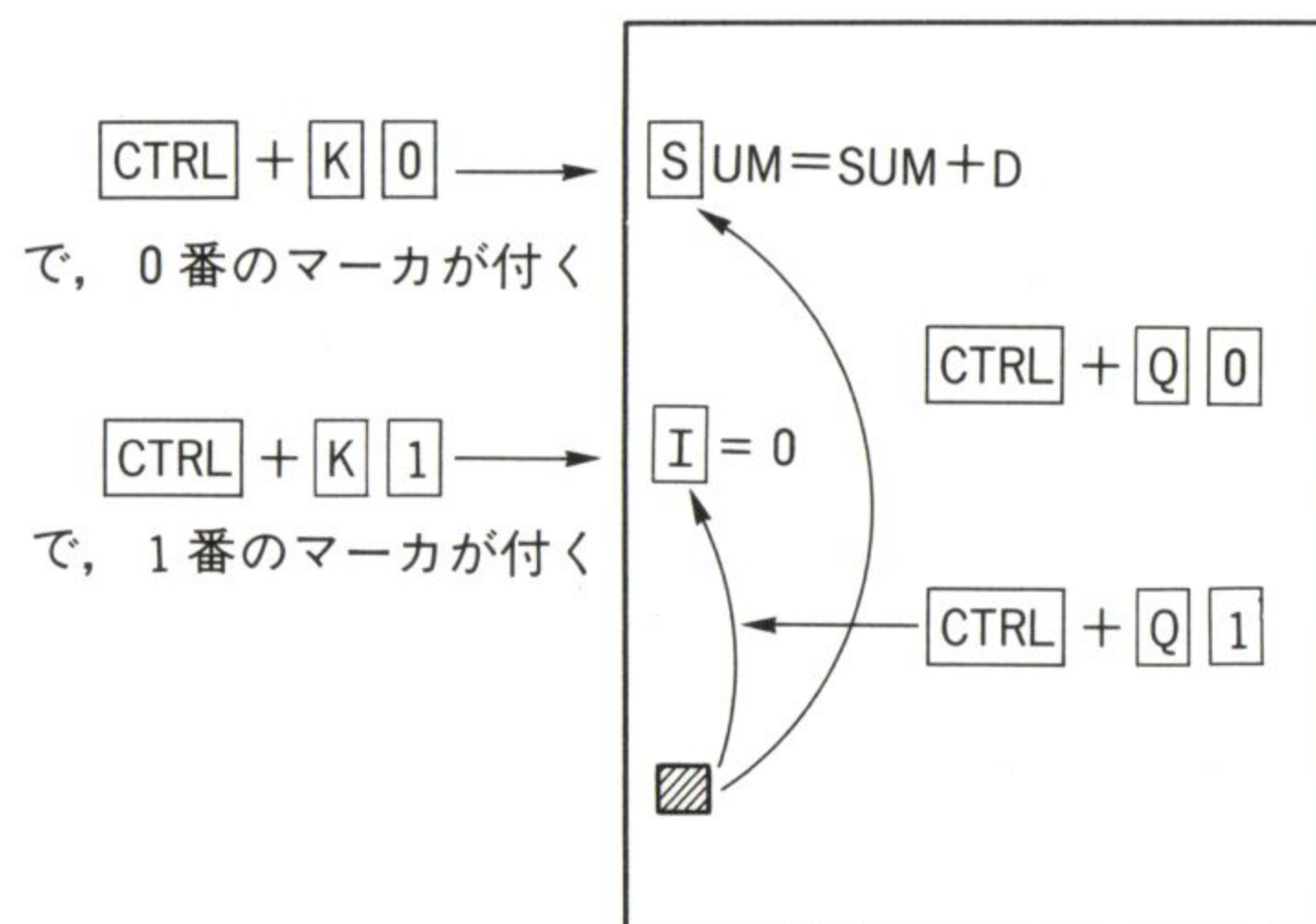
[CTRL]+[K]に続いて[0]～[3]の数字を押すことにより，現在のカーソル行に0～3で認識されるプレース・マーカを設定できます.

逆に[CTRL]+[Q]に続いて[0]～[3]の数字を押すことにより，0～3で認識されるプレース・マーカが設定されている行にカーソルを移すことができます.

[CTRL]+[K][0]
で，0番のマーカが付く

[CTRL]+[K][1]
で，1番のマーカが付く

SUM=SUM+D
I=0

[CTRL]+[Q][0]
[CTRL]+[Q][1]

■アン・ドゥ機能

修正したくないテキストを誤って修正してしまったような場合は，カーソルがその行に残っているなら，

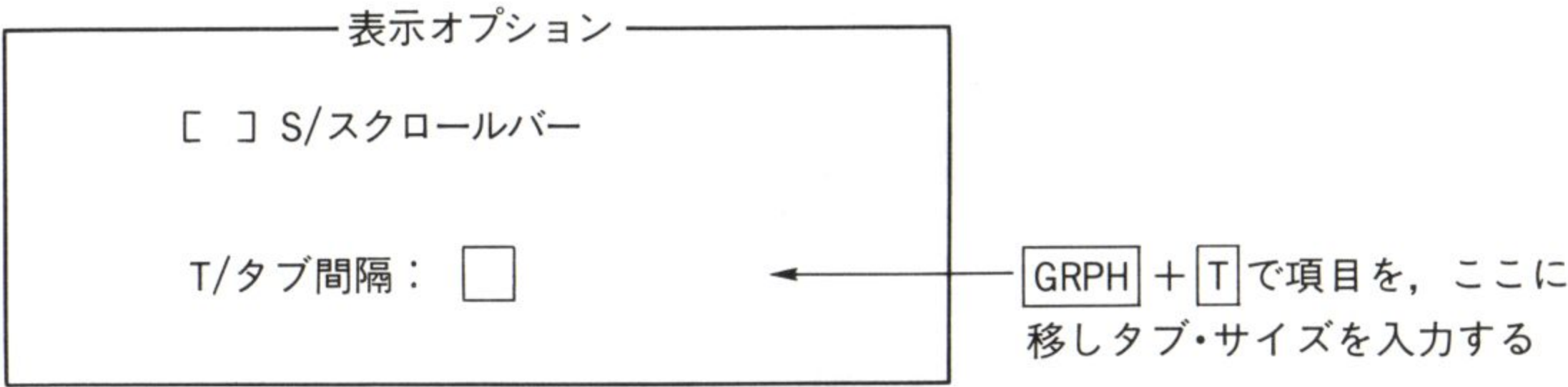
GRPH + BS

により修正を行う直前の内容に修復することができます．これをアン・ドゥ機能と呼びます．

ただし，CTRL + Y による1行削除では，カーソルはその行に残らないので GRPH + BS では修復できません．SHIFT + INS より修復してください．

■タブ・サイズの変更

Quick BASIC でのデフォルトのタブ・サイズは4文字に設定されていますが，V/表示 — O/オプション... メニューにより変更することができます．

```
┌──────── 表示オプション ────────┐
│                                  │
│     [ ] S/スクロールバー         │
│                                  │
│     T/タブ間隔：[ ]     ←────── GRPH + T で項目を，ここに
│                                  │    移しタブ・サイズを入力する
└──────────────────────────────────┘
```

注）Ver.4.5では，O/オプション — D/画面表示 で指定します．

■1行につなげる方法

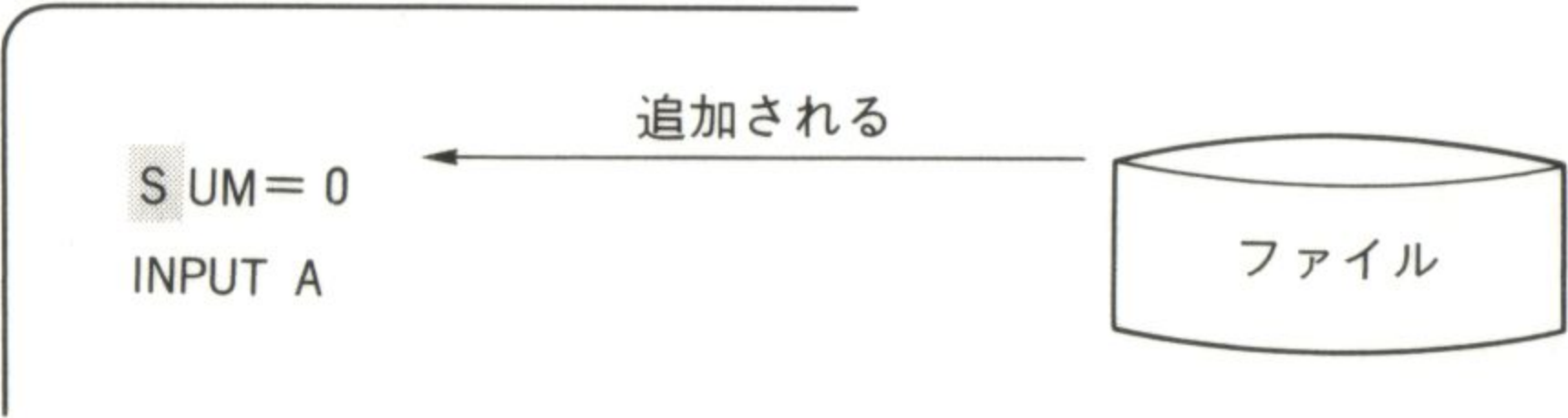
行の途中で ↵ を入力すると，カーソル位置の前後で次のように2行に分かれてしまいます．

```
FOR  k = 1  TO  10
        ⇓ ↵
FOR
k = 1  TO  10
```

これを1行に戻すには，カーソルを下の行の先頭に移して BS を押すか，カーソルを上の行のテキストの後端に移し DEL を押します．

■ほかのファイルをテキスト内に追加する方法

F/ファイル — M/追加... により，ほかのファイルを現在作業中のテキストのカーソル位置の上部に挿入することができます．

SUM＝0
INPUT A
追加される
ファイル

■画面表示色のカスタマイズ

V/表示 — O/オプション により，画面表示色を変更することができます．
注）Ver.4.5では，O/オプション — D/画面表示 で指定します．

表示属性

	文字	背景	
N/テキスト	W/白色	B/黒色	[ ] H/反転
C/カレントステートメント	G/緑色	B/黒色	[ ] H/反転
B/ブレークポイント行	R/赤色	B/黒色	[X] H/反転

各ボックスに TAB キーでカーソルを移し，B…黒 W…白までのイニシャルを入力する

X印があれば反転表示属性がON

■プロシージャのウィンドウを開く

編集画面上で「SUB プロシージャ名( ) ↵」と入力すると，自動的にプロシージャ・ウィンドウが開きますが，E/編集 — S/新規 SUB...，E/編集 — F/新規 FUNCTION... を用いても行えます．

2-6 プログラムの実行

1 インタプリタによるプログラムの実行

編集中のプログラムをインタプリタ(QB)を用いて実行するには,

R/実行 ― S/スタート

または,

SHIFT ＋ F･5

で行います.

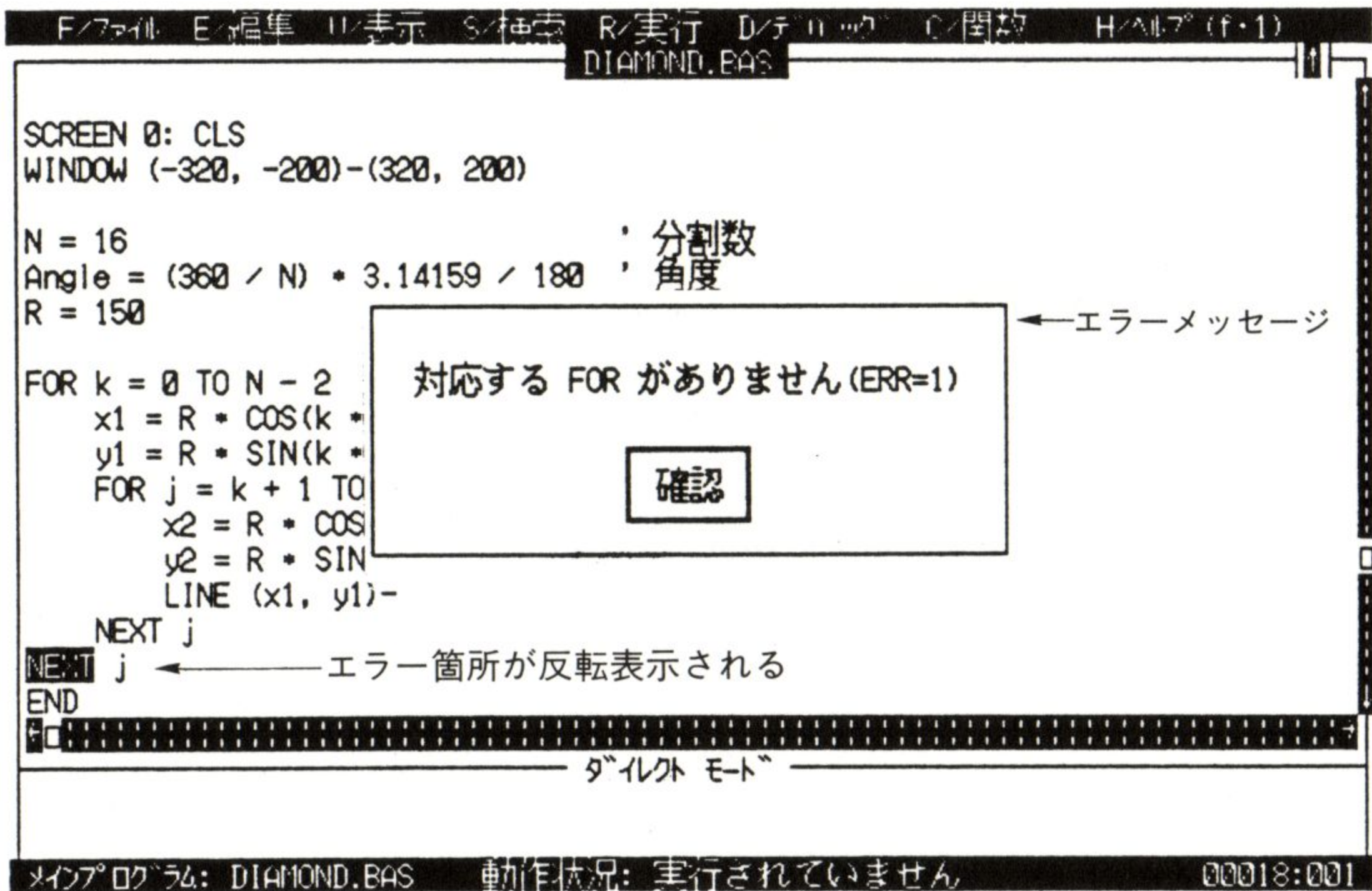
これでインタプリタが文法チェックを行い，誤りがあれば次のようなエラーメッセージを出します.

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 H/ヘルプ(f･1)
DIAMOND.BAS
SCREEN 0: CLS
WINDOW (-320, -200)-(320, 200)

N = 16                                ' 分割数
Angle = (360 / N) * 3.14159 / 180     ' 角度
R = 150

FOR k = 0 TO N - 2
    x1 = R * COS(k *
    y1 = R * SIN(k *
    FOR j = k + 1 TO
        x2 = R * COS
        y2 = R * SIN
        LINE (x1, y1)-
    NEXT j
NEXT j
END
ダイレクト モード
メインプログラム: DIAMOND.BAS   動作状況: 実行されていません   00018:001
```

対応する FOR がありません(ERR=1)

確認

←エラーメッセージ

←エラー箇所が反転表示される

↵または ESC でエラーメッセージ･ボックスが消え，エラー箇所にカーソルが移りますので，誤りを修正して再実行します.

2 実行可能ファイル(.EXE)の作成

インタプリタは，プログラムを開発していく上では対話型でできるため便利ですが，一度完成したプログラムを保存しておく場合にはソースプログラムのレベルでしかできません．したがって，完成したプログラムを実行させる場合には，いちいちインタプリタ(QB)を起動してその上でプログラムを実行しなければならないという，非効率的な面が出てしまいます．

■コンパイラの起動

Quick BASIC では，QB 上から BASIC コンパイラ(BC.EXE, LINK.EXE)を呼び出し，メモリ上のソースプログラムをコンパイル／リンクして実行可能ファイル(.EXE)を作成することができます．

DIAMOND.BAS というプログラムを，QB 上からコンパイル/リンクする方法を以下に示します．

R/実行
X/EXE ファイル作成...

ダイアログ・ボックス

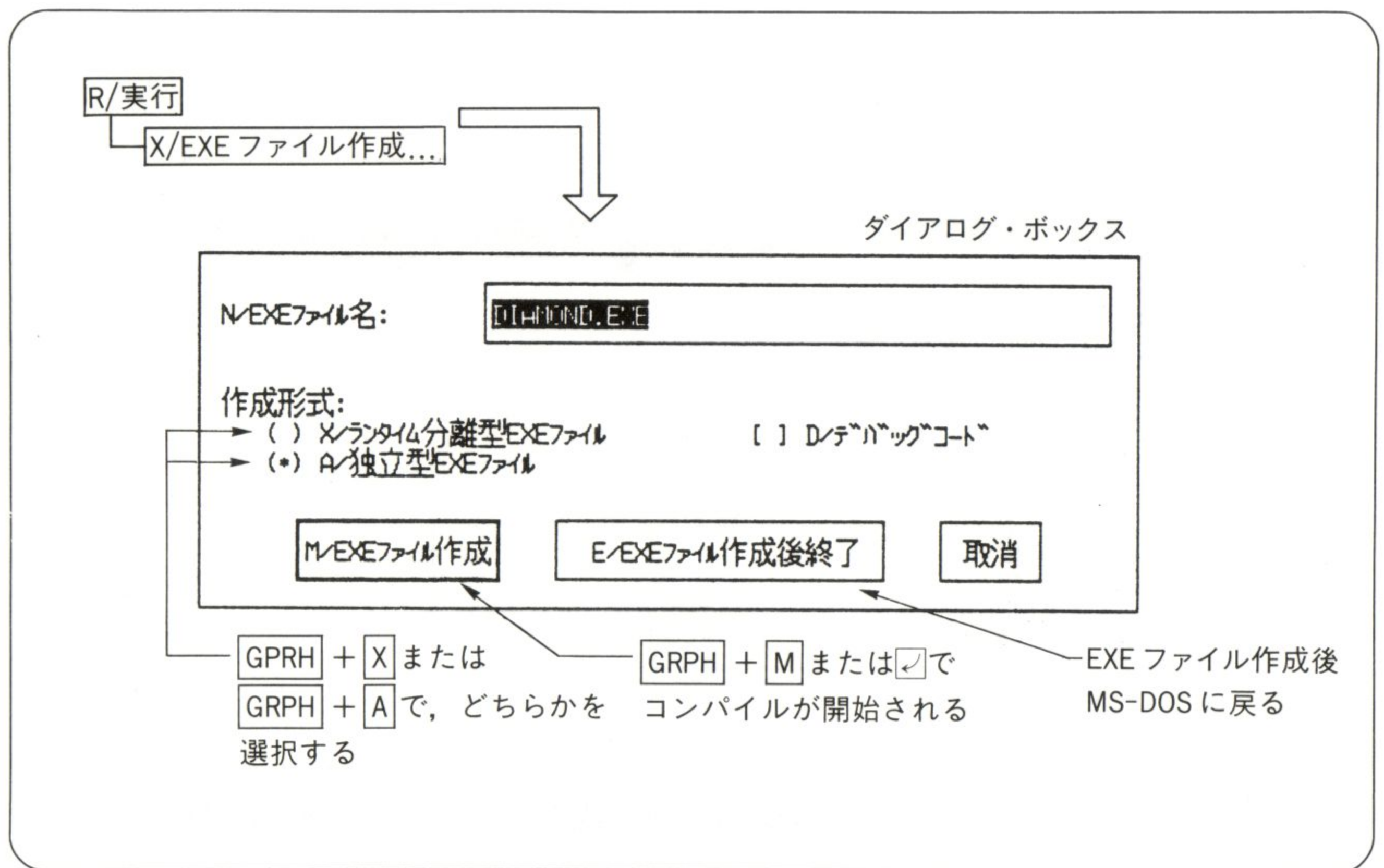
N/EXEファイル名: DIAMOND.EXE

作成形式:
( ) X/ランタイム分離型EXEファイル
(•) A/独立型EXEファイル
[ ] D/デバッグコード

M/EXEファイル作成　E/EXEファイル作成後終了　取消

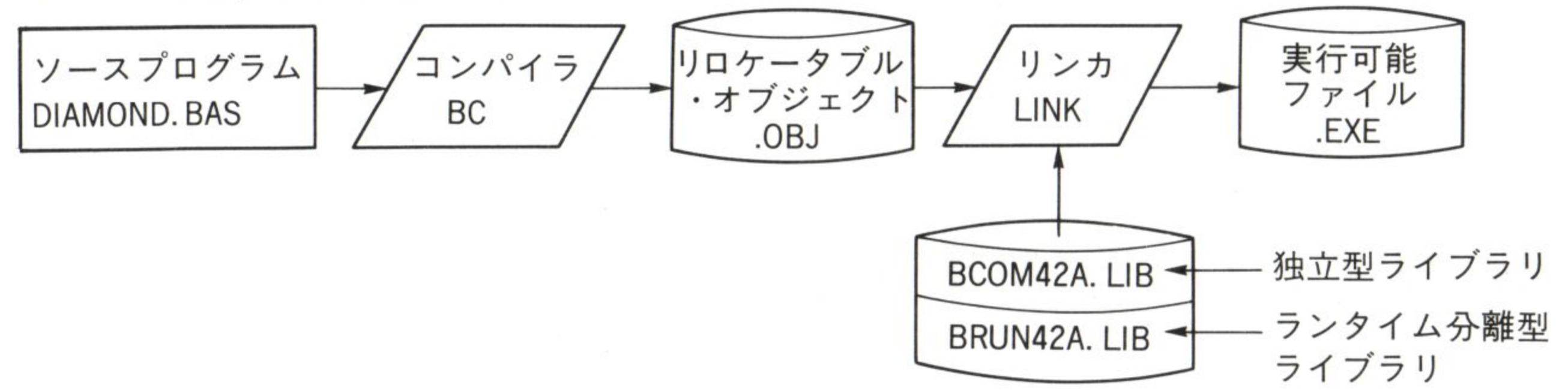
GPRH + X または GRPH + A で，どちらかを選択する

GRPH + M または ↵ でコンパイルが開始される

EXE ファイル作成後 MS-DOS に戻る

上の操作で次のようにコンパイル/リンクが開始され，ドライブ B に DIAMOND.EXE という実行可能ファイルが作成されます．

・QBからコンパイラ(BC.EXE,LINK.EXE)を起動した様子

```
BC B:¥DIAMOND.BAS/T/C/:512;
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20
(C) Copyright Microsoft Corporation 1982-1988.
All rights reserved.

43527 Bytes Available
43525 Bytes Free

    0 Warning Error(s)
    0 Severe  Error(s)
LINK /EX DIAMOND,A:¥DIAMOND.EXE,NUL,;

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.
```

・コンパイル/リンクの流れ

実行可能ファイルは，コマンドラインから，

B>DIAMOND [↵]

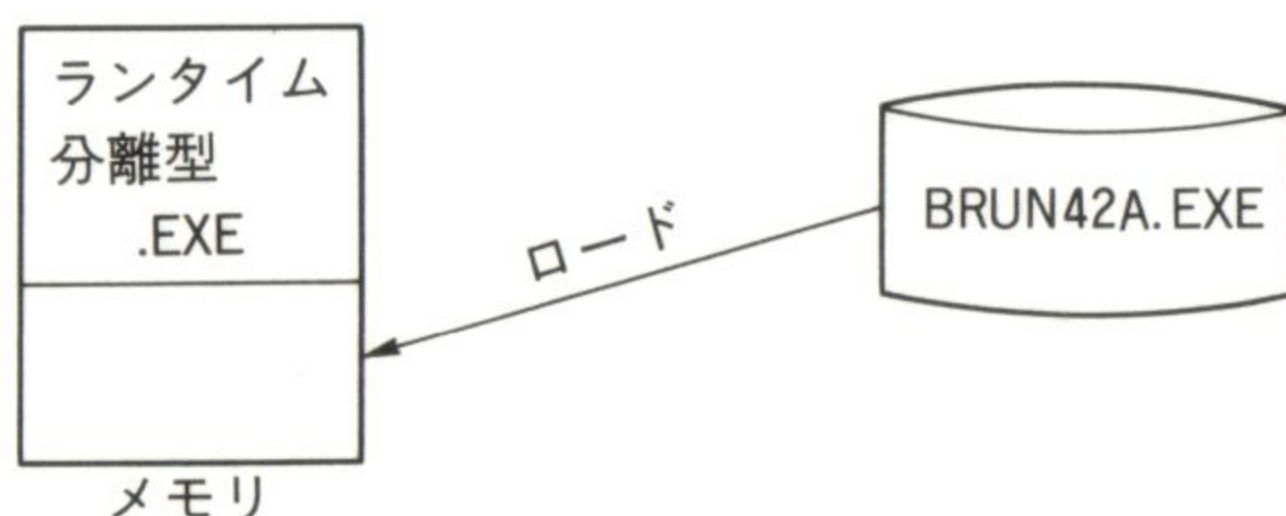
とすることで実行できます．

なお，D/デバッグコード(前頁の[R/実行]―[X/EXE ファイル作成...])を指定しておくと，EXE ファイル中に実行時のエラー処理対処のためのコードが埋め込まれます．ただし，このオプションを指定した.EXE ファイルはCode View デバッガとの互換性はありません．

■実行可能ファイルの作成形式

実行可能ファイルの作成形式は，ランタイム分離型と独立型の2種類が指定できます．使用するライブラリについては，第1章1-2の2の「■ライブラリの選択」(20ページ)を参照してください．

◆**ランタイム分離型**

リンク時に完全な(実体のある)ライブラリを組み込まずに，プログラム実行時にBRUN42A.EXEというランタイム・ライブラリをメモリ上にロードして，これと協力し合って実行を行うタイプの.EXEファイルです．

ランタイム分離型の.EXEはファイル・サイズを小さくすることができますが，実行時に必ずランタイム・ライブラリ BRUN42A.EXEがなければなりません．

◆**独立型**

リンク時に完全なライブラリ(BCOM42A.LIB)を組み込んだ.EXEファイルです．

独立時の.EXEは，ランタイム分離型の.EXEに比べてファイルサイズは大きくなりますが，単独で(BRUN42A.EXEの協力なしで)実行することができます．

3 クィック・ライブラリの作成

利用価値の高いプログラム(モジュール)を共通の資源として利用できるようにしたものをライブラリといいます．

Quick BASICでは，

- **スタンドアロン・ライブラリ(.LIB)**
- **クィック・ライブラリ(.QLB)**

という2種類のライブラリを扱うことができます．

QB統合環境に組み込まれるライブラリをクィック・ライブラリといいます．これに対し，独立して使用されるライブラリをスタンドアロン・ライブラリといいます．スタンドアロン・ライブラリについては，第3章3-3で説明します．

■クイック・ライブラリの作成

次のプログラムは，グラフィック画面のハードコピーを行うものです.

```
DECLARE SUB Gcopy ()
DECLARE SUB Lputc (c%)
DECLARE SUB Lputs (a$)

SUB Gcopy        ' グラフィック画面のハードコピー
    Lputs CHR$(&H1B) + "T16"         ' 改行幅
    Lputs CHR$(&H1B) + "D"           ' コピーモード
    FOR x = &H4F TO 0 STEP -1
        Lputs CHR$(&H1B) + "S0400"   ' 8ビットドット対応
        FOR y = 0 TO 399
            DEF SEG = &HA800: a = PEEK(x + y * &H50)    ' Blue
            DEF SEG = &HB000: b = PEEK(x + y * &H50)    ' Red
            DEF SEG = &HB800: c = PEEK(x + y * &H50)    ' Green
            Lputc (a OR b OR c)
        NEXT y
        Lputc &HD: Lputc &HA         ' 改行
    NEXT x
    Lputs CHR$(&H1B) + "A"           ' ノーマル改行幅
    Lputs CHR$(&H1B) + "M"           ' 通常モード
END SUB

SUB Lputc (c%)       ' 直接プリンタ出力
    WHILE (INP(&H42) AND 4) <> 4
    WEND
    OUT &H40, c%
    OUT &H46, 14
    OUT &H46, 15
END SUB

SUB Lputs (a$)       ' 直接文字列出力
    FOR k = 1 TO LEN(a$)
        c% = ASC(MID$(a$, k, 1))
        Lputc c%
    NEXT k
END SUB
```

このプログラムを，クイック・ライブラリ(GCOPY.QLB)として作成するには，[R/実行]—[L/ライブラリ作成...]を用いて次のようにします.

R/実行
└ L/ライブラリ作成...

N/Quickライブラリファイル名:　gcopy

[ ] D/デバッグコード

[M/ライブラリ作成]　[E/ライブラリ作成後終了]　[取消]

これで次のようにBC(コンパイラ), LINK(リンカ), LIB(ライブラリ・マネジャ)が呼び出され, クィック・ライブラリが作成されます.

```
BC B:¥GCOPY.BAS/T/C/:512;
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20
(C) Copyright Microsoft Corporation 1982-1988.
All rights reserved.

43527 Bytes Available
41633 Bytes Free

    0 Warning Error(s)
    0 Severe  Error(s)
LINK /QU GCOPY,B:¥GCOPY.QLB,NUL,BQLB42.LIB;

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

LIB B:¥GCOPY.LIB+GCOPY;

Microsoft (R) Library Manager  Version 3.08
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.
```

■クィック・ライブラリの利用法

こうして作成したクィック・ライブラリ(GCOPY.QLB)を利用するには, QB起動時に次のように/Lオプションで指定します.

A>QB␣/LGCOPY ↵

これでライブラリがQBシステムに組み込まれますので, ユーザ・プログラムでGCOPYプロシージャを使用するには,

CALL Gcopy

とします. なお, ユーザ・プログラムにおいて,

DECLARE SUB Gcopy( )

と宣言しておけば, CALLをつけずに,

Gcopy

だけで, 呼ぶことができます.

なお, QB起動時の/Lオプションにクィック・ライブラリ名を指定しなければ, QB.QLBがデフォルトで選択されます.

4 コマンド・ライン引数の取得

MS-DOS の COPY コマンドで，ファイル a.txt をドライブ B に b.txt としてコピーするには，

A>copy a.txt b : b.txt ↵

とします．このようにコマンドを投入した行を，コマンド・ラインと呼び，コマンド・ラインからコマンドや実行プログラムに渡されるパラメータを，コマンド・ライン引数と呼びます．

Quick BASIC では COMMAND$ という関数により，コマンド・ライン引数を取得することができます．

次のプログラムは，

A>LIST filename ↵

のようにコマンドを与え，filename の内容をタイプするものです．この場合，プログラム中の COMMAND$ にコマンド・ライン引数の filename が取得されます．

統合開発環境の QB 上でこのコマンド・ライン引数を与えるには，[R/実行]―[C/COMMAND$入力...]を用います．

次のプログラムは，コマンド・ラインから与えられるファイル名のファイルの内容を１行単位で表示していくものです．

・自作のタイプ・コマンド

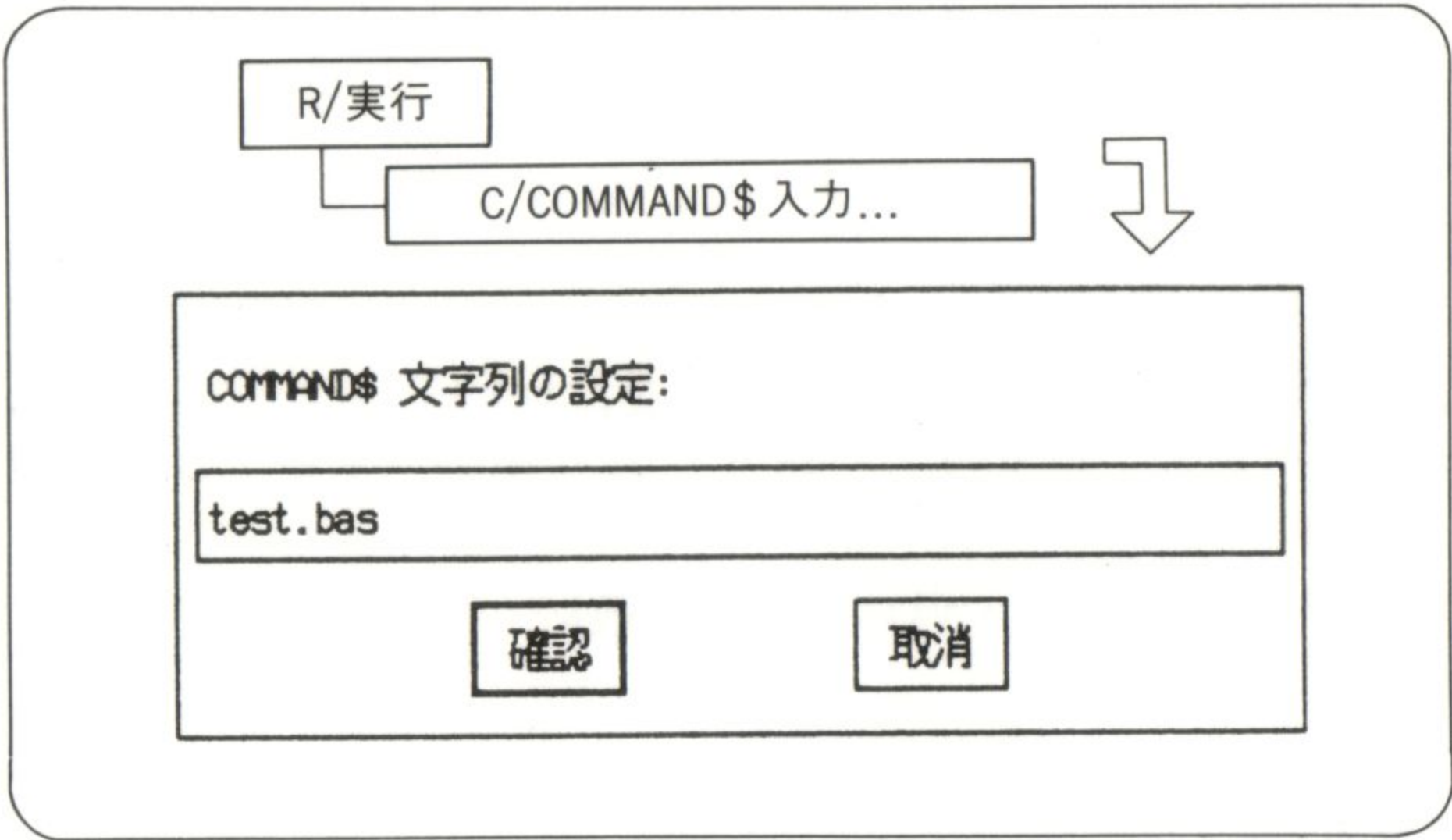
```
' タイプ・コマンド(A>LIST filename)

OPEN COMMAND$ FOR INPUT AS #1
WHILE NOT EOF(1)
    LINE INPUT #1, a$
    PRINT a$
WEND
CLOSE #1
```

このプログラムを QB 上で実行するには，R/実行 ─ C/COMMAND$入力... を用いてあらかじめファイル名を設定しておきます．

R/実行
C/COMMAND$ 入力...

COMMAND$ 文字列の設定:

test.bas

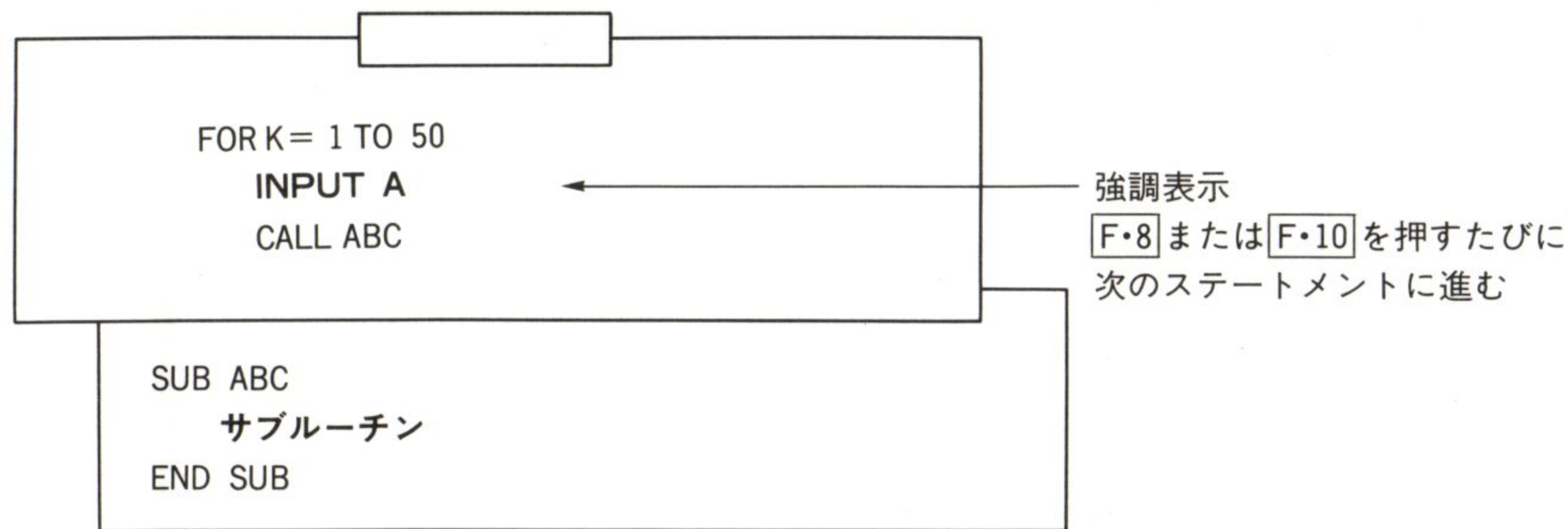
確認　取消

2-7 デバッグ

Quick BASICには，トレース，ウォッチ，ブレーク・ポイントなどのデバッグ機能があります．

1 トレース機能

トレース機能とは，操作者の指令(F･8またはF･10)によりプログラムを1ステートメントずつ実行していく機能です．Quick BASICでは，実行(トレース)されているステートメントが強調表示(緑色)されます．

```
FOR K= 1 TO 50
  INPUT A        ← 強調表示
  CALL ABC

SUB ABC
  サブルーチン
END SUB
```

強調表示
F･8またはF･10を押すたびに
次のステートメントに進む

F･8とF･10の違いは，呼び出したサブルーチン(プロシージャ)の内部もトレースするか否かの点です．F･8は呼び出したサブルーチン(上の例ではサブルーチンABC)の内部もトレースしますが，F･10では行いません．

2 プログラムの低速実行

D/デバッグ—T/トレースを選択してトレース・モードをONにしておき，SHIFT＋F･5でプログラムを実行すると，1ステートメント単位でゆっくりと行われます．実行中のステートメントは強調表示されますので，プログラムの全体的な流れを見るのに便利です．後述するウォッチ機能と組み合わせて使うと効果的です．

3 ウォッチ機能

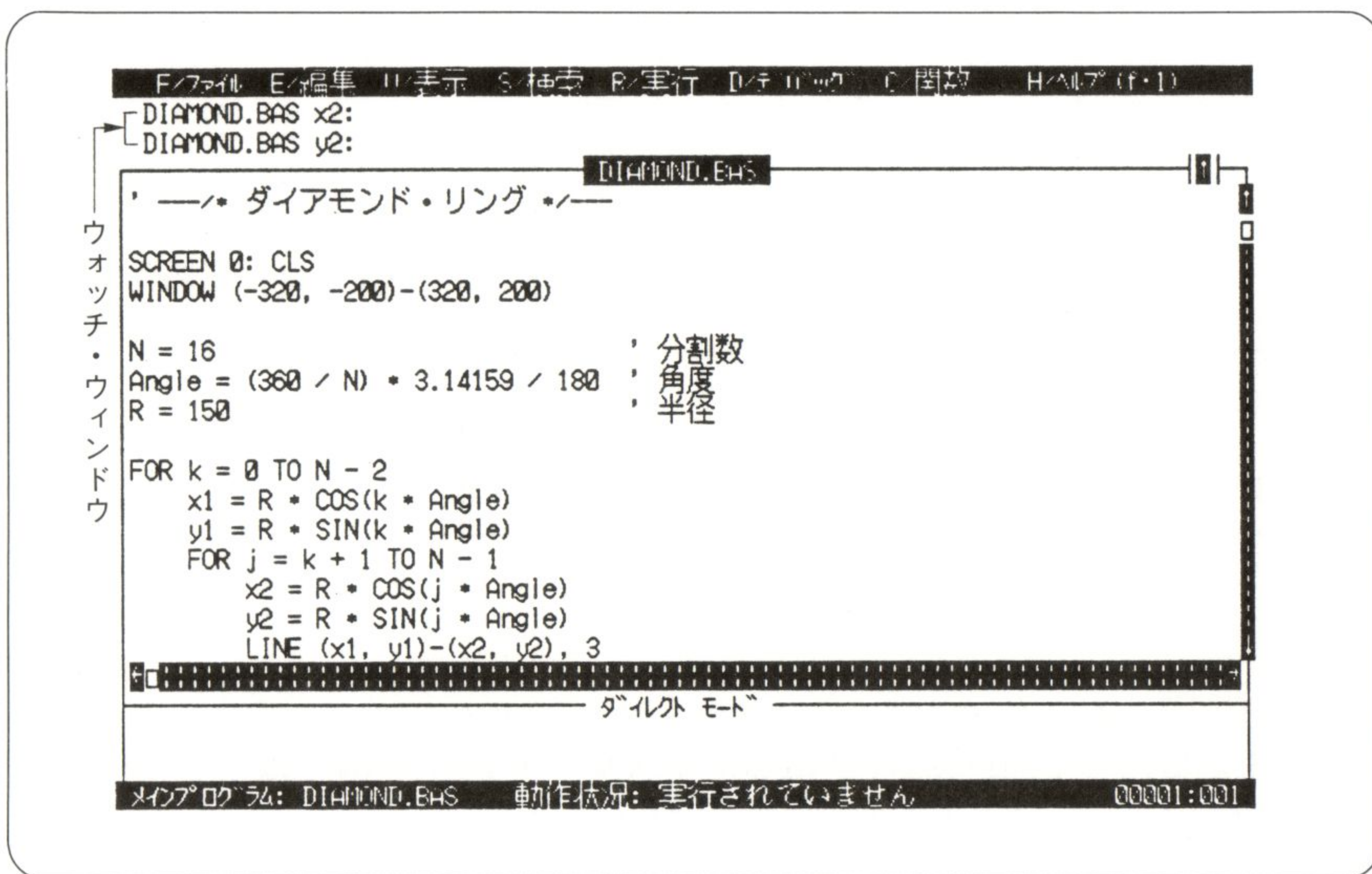
プログラム実行中の変数や式の値を，ウォッチ･ウィンドウに刻々と表示する機能をウォッチ機能といいます．

たとえば，変数x2の値をウォッチしたい場合は，[D/デバッグ]—[A/ウォッチの追加...]を選択し，次のようにウォッチ式(変数，変数を含む式など)を入力します．

```
ウォッチウィンドウに登録する式・変数の設定：
x2 ←——ウォッチ式を入力する
```

変数y2についても同様な方法で指定します．

これで次のようにウォッチ･ウィンドウにウォッチ式が登録されます．

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 H/ヘルプ (f･1)
DIAMOND.BAS x2:
DIAMOND.BAS y2:
DIAMOND.BAS
' ——/* ダイアモンド・リング */——

SCREEN 0: CLS
WINDOW (-320, -200)-(320, 200)

N = 16                                   ' 分割数
Angle = (360 / N) * 3.14159 / 180        ' 角度
R = 150                                  ' 半径

FOR k = 0 TO N - 2
    x1 = R * COS(k * Angle)
    y1 = R * SIN(k * Angle)
    FOR j = k + 1 TO N - 1
        x2 = R * COS(j * Angle)
        y2 = R * SIN(j * Angle)
        LINE (x1, y1)-(x2, y2), 3
ダイレクト モード
メインプログラム: DIAMOND.BAS   動作状況: 実行されていません   00001:001
```

ウォッチ・ウィンドウ

このウォッチ式を削除する場合には，[D/デバッグ]—[D/ウォッチの削除...]または[D/デバッグ]—[L/全ウォッチの削除]を用います．

[F･8]，[F･10]でプログラム･トレースすると，ウォッチ･ウィンドウのウォッチ式の値がプログラムの進行に応じて変化します．

4 ブレーク・ポイント

ブレーク・ポイントとは，プログラムの実行を一時中断する位置(行)のことです．プログラムをデバッグするときに，プログラムの途中経過を見るために止めたい場所があります．こうしたときに，この位置にブレーク・ポイントを設定してからプログラムを実行すると，プログラムの実行がブレーク・ポイントにきたときに中断されます．このとき，ダイレクト・モードで変数の内容を調べるなどのデバッグ作業が行えます．

ブレーク・ポイントの設定は，設定したい位置(行)にカーソルを移動し，

F･9

を押します．これで，この行が赤色のバーで表示されます．すでにブレーク・ポイントが設定されている行で再びF･9を押すと，ブレーク・ポイントの解除になります．

ブレーク・ポイントは複数設定することができます．設定したブレーク・ポイントをすべて解除するには，D/デバッグ—C/全ブレーク・ポイントの解放を用います．

5 ウォッチ・ポイント

ウォッチ・ポイントは条件付きのブレーク・ポイントと考えられます．D/デバッグ—W/ウォッチポイント...で設定した条件式が真(TRUE)になったときに，プログラムが中断されます．

D/デバッグ
W/ウォッチポイント...

条件式の設定:(式=真になるまで実行します)

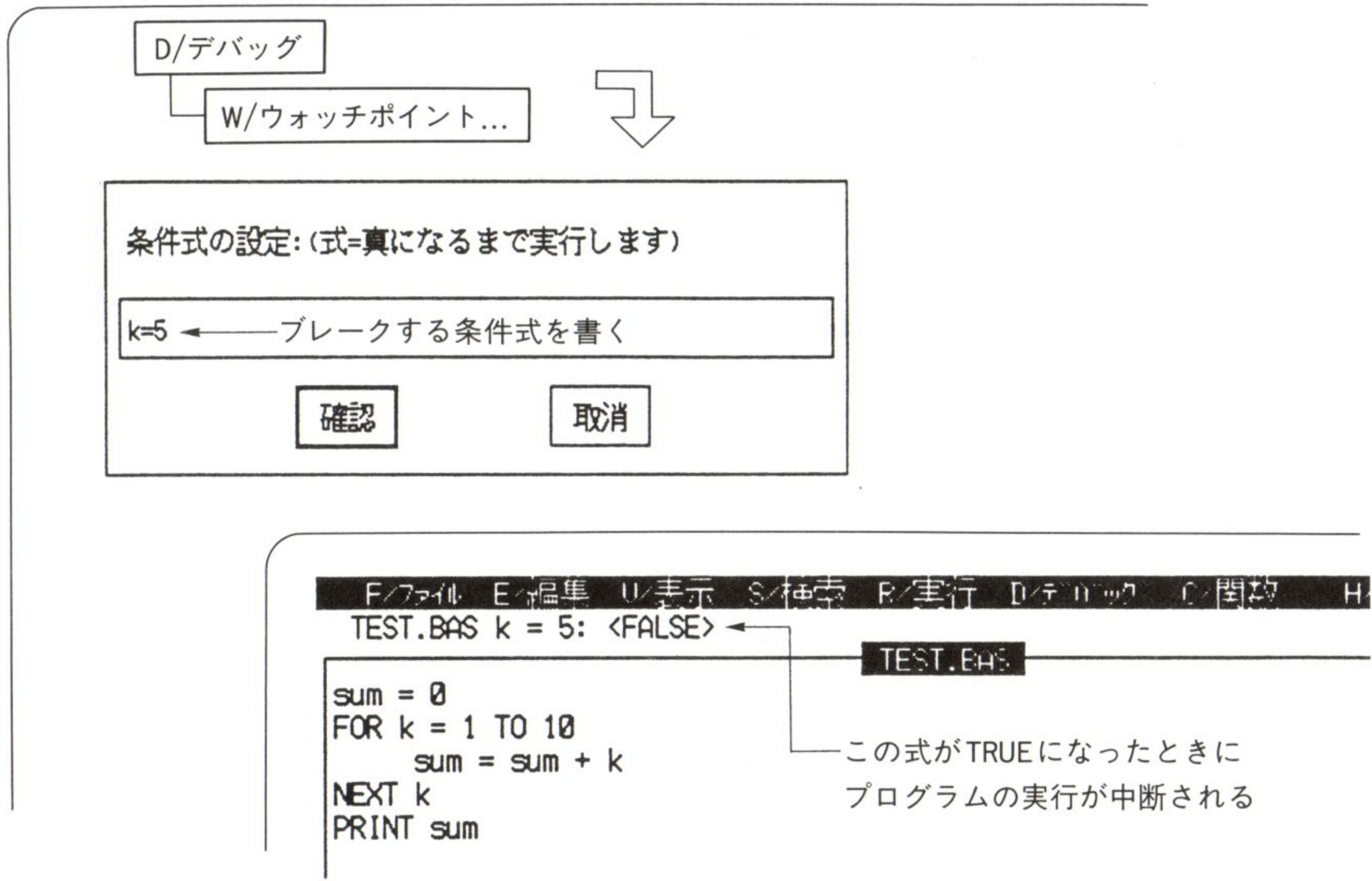
k=5 ←ブレークする条件式を書く

確認　取消

F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 H/

TEST.BAS k = 5: <FALSE> ←この式がTRUEになったときにプログラムの実行が中断される

TEST.BAS

```
sum = 0
FOR k = 1 TO 10
      sum = sum + k
NEXT k
PRINT sum
```

6 ダイレクトモード

F·6でアクティブ·ウィンドウをダイレクトモード·ウィンドウに移すことで，ダイレクトモードでの仕事を行うことができます.

ダイレクトモードとは，BASICのコマンドおよびステートメントを書き，↵を入力することで，そのコマンドおよびステートメントを即座に実行するものです．ダイレクトモードはおもにデバッグ作業で使います.

たとえば，変数x2，y2の内容を調べたいときは，

ダイレクトモード

```
?  x2,y2 ↵
```

とします.

ダイレクトモードは1行に収まる範囲で書きます．次のようなマルチステートメント(複数のステートメントを：で区切って1行にしたもの)を書くことができます.

```
for k=0 to 10 : ? a(k) : next k ↵
```

これで配列a(0)～a(10)の内容が表示されます.

プログラムを中断し，ダイレクトモードで，プログラム中で使用している変数の値を変更した場合，プログラムを再開したときの変数の値は，変更された値となります.

ダイレクトモードから次のようにプロシージャを呼び出すこともできます.

```
call disp
```

ダイレクトモード·ウィンドウには最大10行まで入力できますが，それぞれの行は互いに独立しています．ダイレクトモード·ウィンドウを拡大するにはCTRL＋F·10をくり返して押します.

7 実行の制御

デバッグ中のプログラムは，プログラムの実行，中断，継続がくり返し行われます．これを制御するために次のメニュー(キー)を用います．

[R/実行]—[R/リスタート]

プログラム中のすべての変数をクリアして，プログラムの最初の実行可能ステートメントを実行開始位置に設定します．トレース作業を最初からやり直すときなどに使います．

[R/実行]—[N/続行]　([F・5])

ブレーク・ポイント，STOP 文などにより中断されたプログラムを中断位置から再開します．

[F・7]

現在プログラムが実行されている行から，カーソルが置かれている行まで実行します．

[D/デバッグ]—[S/カレントステートメント]

次に実行される行を，カーソルが置かれている行に設定します．これは，現在の実行行からカーソル行までを実行していくのではなく，その間をスキップします．

[V/表示]—[N/次のステートメント]

プログラムを中断し，デバッグ作業によりカーソルがほかの行に移ってしまったときに，このメニューにより，プログラムの実行を継続する際の実行行にカーソルを移します．

8 ヒストリ

[D/デバッグ]—[H/ヒストリ]を ON にすることで，QB は自動的に，実行されたプログラムの最後の20行を記録します．この実行記録をヒストリと呼びます．

このヒストリは，次のキーにより追跡することができます．

[SHIFT]＋[F・8]

最後に実行された行から，20行前の実行行までを1行ずつカーソルを逆に戻していきます．

[SHIFT]＋[F・10]

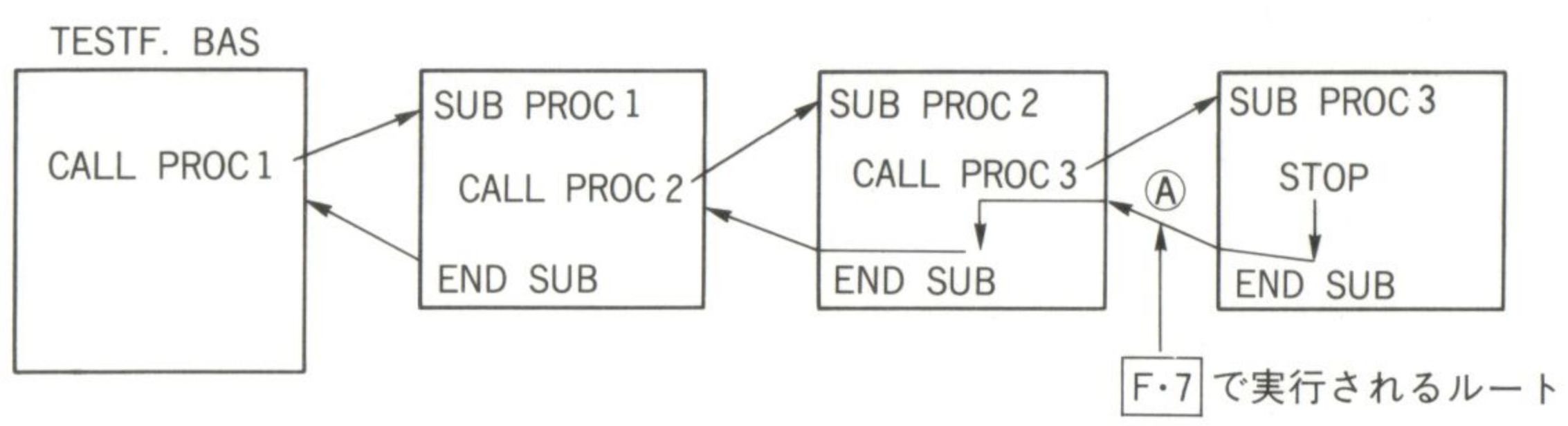
20行前の実行行から最後の実行行に向かってカーソルを1行ずつ移していきます．

9 プロシージャ・コールの追跡

たとえば次のように，プロシージャを次々にネストして呼び出しているようなプログラムがあったとします．

TESTF. BAS

TESTF. BAS			
CALL PROC1	SUB PROC1 CALL PROC2 END SUB	SUB PROC2 CALL PROC3 END SUB	SUB PROC3 STOP END SUB

Ⓐ F・7 で実行されるルート

このプログラムを実行すると，PROC3のSTOP文で実行が中断されます．このとき C/関数 により，次のようなプロシージャのコール順序の一覧が表示されます．一覧表の上になるほど最新に呼び出されたプロシージャです．

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数   H/ヘルプ (f・1)
┌──────────────── TESTF.BAS:proc3 ───  ┌──────────────┐
SUB proc3                               │proc3         │
    PRINT "proc3"                       │proc2         │
    STOP                                │proc1         │
    PRINT "owari3"                      │/TESTF.BAS    │
END SUB                                 └──────────────┘
```

一覧表のプロシージャ名にカーソルを移し，↵を押すと，そのプロシージャがアクティブ・ウィンドウに表示されます．さらに，F・7を押せば，STOPなどで中断した位置からそのプロシージャの箇所まで，プログラムが実行(プロシージャ・コールをリターンしながら)されます．

たとえば，PROC1にカーソルを移し↵を押して，F・7を押した場合は，先のプログラムの図のⒶのルートに沿って，プロシージャPROC1のCALL PROC2の次の実行文の直前までを実行します．

2-8 ヘルプ

1 オンラインヘルプ

Quick BASICには，オンラインヘルプという強力なヘルプ機能があります．プログラム作成中に，たとえばLINEというステートメントの使い方を調べたい場合，いちいちマニュアルを調べなくても，プログラム中のLINEという文字のところにカーソルを移し，

SHIFT + f･1

を押すことで，次のようなヘルプメッセージが表示されます．

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数    H/ヘルプ(f･1)
LINE INPUT[;] ["promptstring";] stringvariable

標準入力先から stringvariable に 1行のデータを読み込みます.
-------------------------------------------------------------------------
LINE INPUT #filenumber,stringvariable

filenumber でオープンしたファイルから1行分のデータを読み込み，その行を string
variable に代入します.
-------------------------------------------------------------------------
LINE [[STEP]((x1,y1)]-[STEP] (x2,y2) [,[color][,[B[F]][,style]]]

線を引いたり，ボックスを描いたりします(グラフィックモード).
------------------------------- DIAMOND.BAS -----------------------------
        y2 = R * SIN(j * Angle)
        LINE (x1, y1)-(x2, y2), 3
    NEXT j
NEXT k
END

------------------------------ ダイレクト モード ------------------------------

メインプログラム: DIAMOND.BAS    動作状況: 実行されていません          00016:009
```

← ヘルプメッセージ

2 全般的なヘルプ

F･1 または H/ヘルプ ─ G/全般... により，次のような全般的なヘルプ情報が表示されます．

ヘルプ画面から抜けるには ESC を押します．

エディットコマンドとデバッグコマンド

挿入
挿入／上書きモードの切り換え INS
現在行の上に1行挿入 HOME CTRL+N
現在行の下に1行挿入 END CTRL+N
クリップボードから挿入 SHIFT+INS

選択
文字列／行 SHIFT+ARROW
単語 SHIFT+CTRL+ARROW

削除
現在行（クリップボードに保管） CTRL+Y
ファイルの終端（クリップボードに保管） CTRL+Q Y
指定文字列（クリップボードに保管） SHIFT+DEL
指定文字列（クリップボードに保管しない） DEL

コピー
指定文字列 CTRL+INS

検索
指定文字列 CTRL+¥
再検索 f･3

デバッグ
出力画面の表示 f･4
カーソルのある行まで実行 f･7
ブレークポイントの設定／削除 f･9
1行ごとの実行 f･8
プロシージャごとの実行 f･10
前方ヒストリの実行 SHIFT+f･10
後方ヒストリの実行 SHIFT+f･8

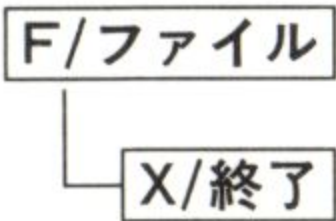
N／次のページ　P／前のページ　K／キーワード　取消

注）Ver.4.5では，QB アドバイザーによりもっと詳細なヘルプが得られます．

2-9 QBモードから抜ける

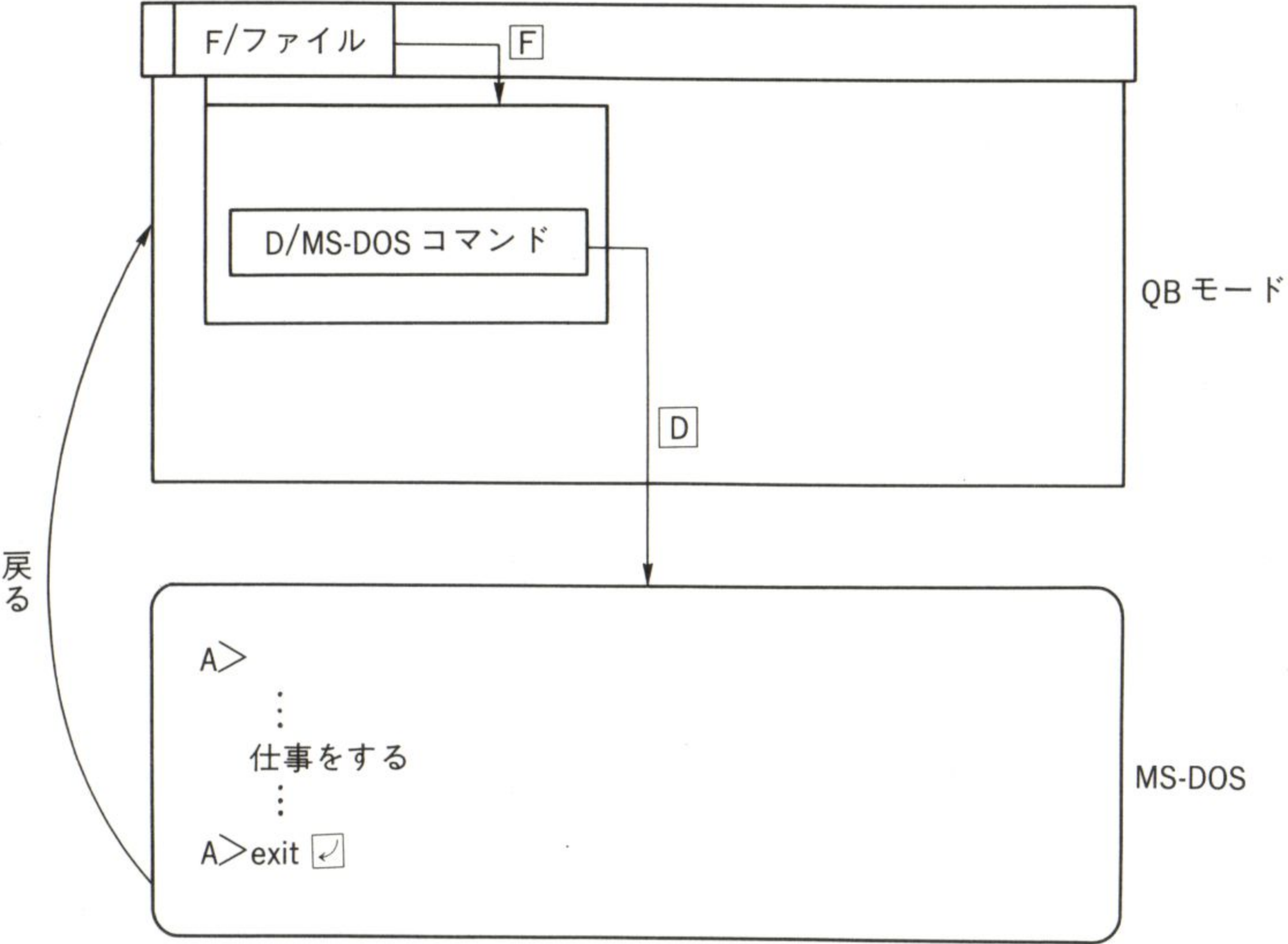
QBモードから抜けてMS-DOSに戻るには，次の2つの方法があります．

1 QBを終了してMS-DOSに戻る

QBモードを終了してMS-DOSに戻るには,次のメニューを選択します．

F/ファイル
└X/終了

この場合，再びQBモードに入るには，QBを再起動しなければなりません．

2 QBからMS-DOSに一時的に戻る(OSシェル)

F/ファイル─D/MS-DOSコマンドを選択することで，QBモードから一時的にMS-DOSに戻ることができます．

MS-DOS 上でいろいろな仕事をしたあとに，

```
A>exit ↵
```

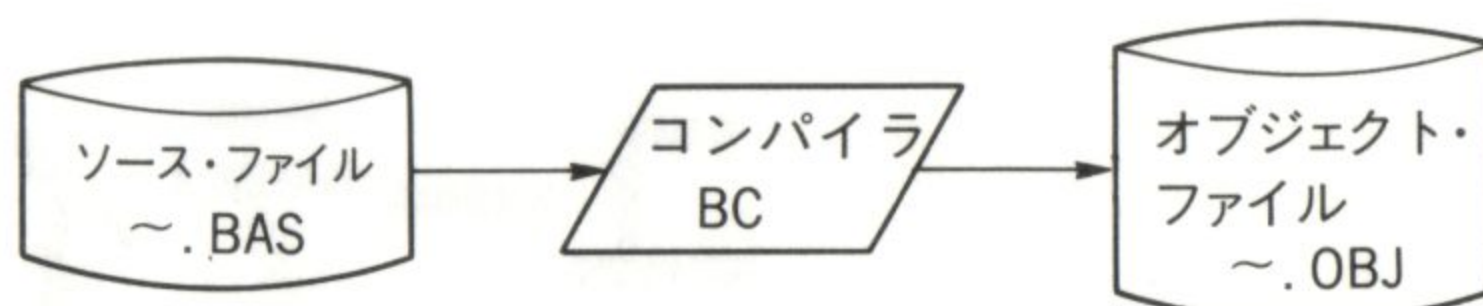
と入力すれば，MS-DOS に移る前の QB モードに瞬時に戻ります．
QB の起動は時間がかかるので，この OS シェル機能はたいへん便利です．

第３章

コマンド・バージョン・ユーティリティ

3−1 ベーシック・コンパイラ(BC)

BCはBASICのソース・ファイル(.BAS)をコンパイルし，リロケータブル・オブジェクト・ファイル(.OBJ)を作成するものです．

BCにかけるソース・ファイルはテキスト・ファイルでなければなりませんので，QBでプログラムを作る場合は「T/テキストファイル」を指定してセーブしておかなければなりません．

1 BCの起動法

BCの起動法には，次の2つの方法があります．

■方法1　単独で起動する

```
A>BC↵

Source Filename [.BAS]: ソース・ファイル名↵
Object Filename [filename.OBJ]: オブジェクト・ファイル名↵
Source Listing   [NUL.LST]: ソース・リスティング・ファイル名↵
```

各プロンプトに対応するファイル名を入力します．↵のみ入力すると，[　]で囲まれたデフォルトのファイル名が付けられます。ファイル名のタイプを省略すると，[　]内のタイプが採用されます．各応答の最後に/で始まるオプションを指定することができます．

プロンプトに対してセミコロン(;)を入力すると，それ以後のプロンプトは表示されず，デフォルトのファイル名が採用されます．

NULとは，ファイルを何も作らないということを意味しています．ファイルはカレント・ドライブのカレント・ディレクトリに作られますので，ほかへ作りたいときはドライブ名またはパス名を付けてください．

ファイル名の大/小文字は区別されません．

例

```
A>bc
Microsoft (R) QuickBASIC KANJI Compiler Version 4.20
(C) Copyright Microsoft Corporation 1982-1988.
All rights reserved.
Source Filename [.BAS]: test
Object Filename [test.OBJ]: test
Source Listing [NUL.LST]:

43511 Bytes Available
43139 Bytes Free

    0 Warning Error(s)
    0 Severe  Error(s)
```

■方法2　ファイル名をコマンド・ラインに与えて起動する

A>BC　ソース・ファイル名，[オブジェクト・ファイル名]，[リスティング・ファイル名][/オプション]

ファイルは方法1の場合と同様のものです．ファイル名を指定せずにカンマ(,)のみを置くと，そのファイル名はデフォルト解釈されます．またセミコロン(;)を置くと，それ以後のファイル名はすべてデフォルト解釈されます．

例

A>BC TEST;↵
…ドライブAのTEST.BASをコンパイルし，TEST.OBJを作る．

A>BC B:TEST,B:;↵
…ドライブBのTEST.BASをコンパイルし，ドライブBにTEST.OBJを作る．

A>BC TEST,,ABC↵
…ドライブAのTEST.BASをコンパイルし，TEST.OBJを作り，さらにABC.LSTも作る．

2 BCのコンパイル・オプション

コンパイル・オプションは，スラッシュ(/)またはハイフン(-)で始まる次のような文字で指定します．オプション文字の大/小は区別されません．

■/A （アセンブリコードリスティングの付加）

リスティング・ファイルに，次のようなアセンブリ言語コードを埋め込みます．

```
                                                              PAGE   1
                                                              15 Aug 87
                                                              19:04:46
Offset  Data     Source LineMicrosoft (R) QuickBASIC KANJI Compiler Version 4.20

  0030  0006     sum = 0
  0030  0006     FOR k = 1 TO 10
  0030   **             I00002: mov   bx,offset <00000000>
  0033   **                     call  __flds
  0038   **                     mov   bx,offset SUM!
  003B   **                     call  __fstsp
  0040   **                     mov   bx,offset <0000803F>
```

■/AH （動的配列に対するメモリの解放）

動的配列に利用可能なすべてのメモリを解放します．このオプションを指定しなければ，１つの配列に対する最大サイズは64KBです．

INPUT N
DIM A(N)　　　…動的配列．/AH を付ければ64KB以上でもメモリオーバ・エラーにならない．
DIM B(30000)　…静的配列．64KB以上はとれない．

■/C:n （通信ポート・バッファのサイズ）

通信ポートの非同期通信アダプタを使用して，リモートデータを受信するためのバッファサイズをnバイトに設定します．デフォルトは256 バイトです．

■/D （ランタイム・エラー用デバッグ・コードの生成）

ランタイム・エラーを調べるための，デバッグ・コードを埋め込みます．これにより STOP を押すと，プログラムの実行をブレークすることができます．

■/E　/X（RESUME ステートメントの使用）

/E は RESUME linenumber ステートメントを使った，ON ERROR ステートメントをプログラムで使用している場合に指定します．

/X は RESUME，RESUME NEXT，RESUME 0 ステートメントを使った ON ERROR ステートメントをプログラムで使用している場合に指定します．

第 5 章の 5 -11を参照してください．

■/FPI　/O　（ライブラリの指定）

ライブラリとして以下の 4 種類のどれを使用するかを指定します．

/FPI はエミュレート・ライブラリの指定，/O はスタンドアロン・ライブラリの指定を行います．

/FPI ↓（8087/80287エミュレート列）

/O →（スタンドアロン型行）

	代替数値演算	8087/80287エミュレート
スタンドアロン型	① BCOM42A.LIB	② BCOM42E.LIB
ランタイム分離型	③ BRUN42A.LIB	④ BRUN42E.LIB

したがって，/O と/FPI の組み合わせは以下の 4 種類となります．

デフォルト	③
/O	①
/FPI	④
/O　/FPI	②

注）Ver.4.5では，明示的に代替数値演算を指定する/FPA が追加されました．

■/MBF　（数値形式の指定）

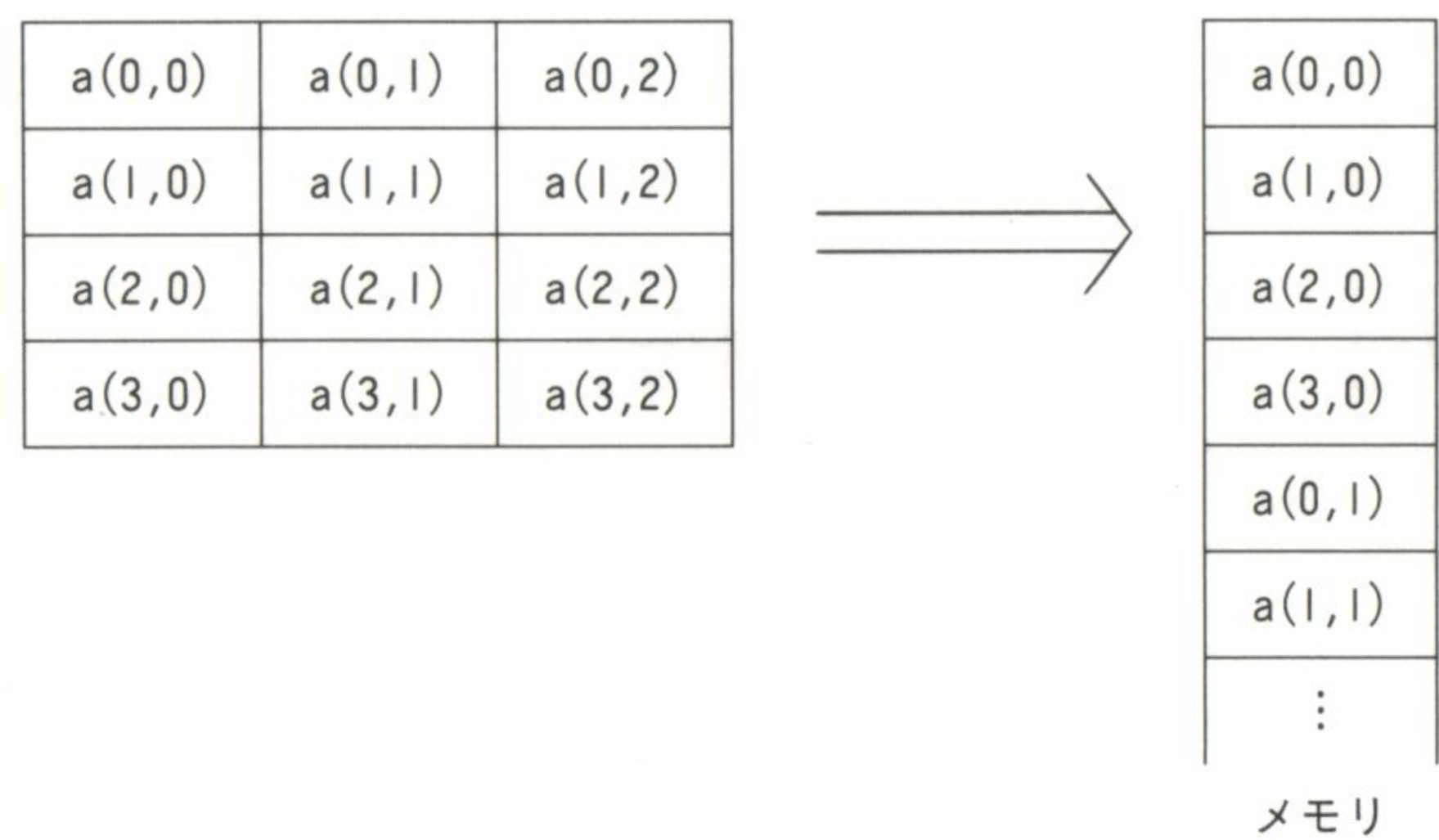
MK~$，CV~などの変換関数に対し，IEEE 形式の数値をマイクロソフトバイナリ形式の数値として扱うようにします．

■/R (配列要素のメモリ格納順序)

配列要素のメモリ格納順序を行方向にします．FORTRAN はこの順序で格納します．

a(0,0)	a(0,1)	a(0,2)
a(1,0)	a(1,1)	a(1,2)
a(2,0)	a(2,1)	a(2,2)
a(3,0)	a(3,1)	a(3,2)

⟹

メモリ
a(0,0)
a(1,0)
a(2,0)
a(3,0)
a(0,1)
a(1,1)
⋮

■/S (引用文字列の埋め込み方法)

引用文字列をシンボルテーブルでなく，オブジェクト・ファイルに書き込みます．

■/V /W (イベント・トラッピングの使用)

通信(COM)，タイマ(TIMER)，ファンクションキー(KEY) などのイベント・トラッピングを可能にします．

/V はステートメント間で発生するイベントをチェックし，/W は行間で発生するイベントをチェックします．

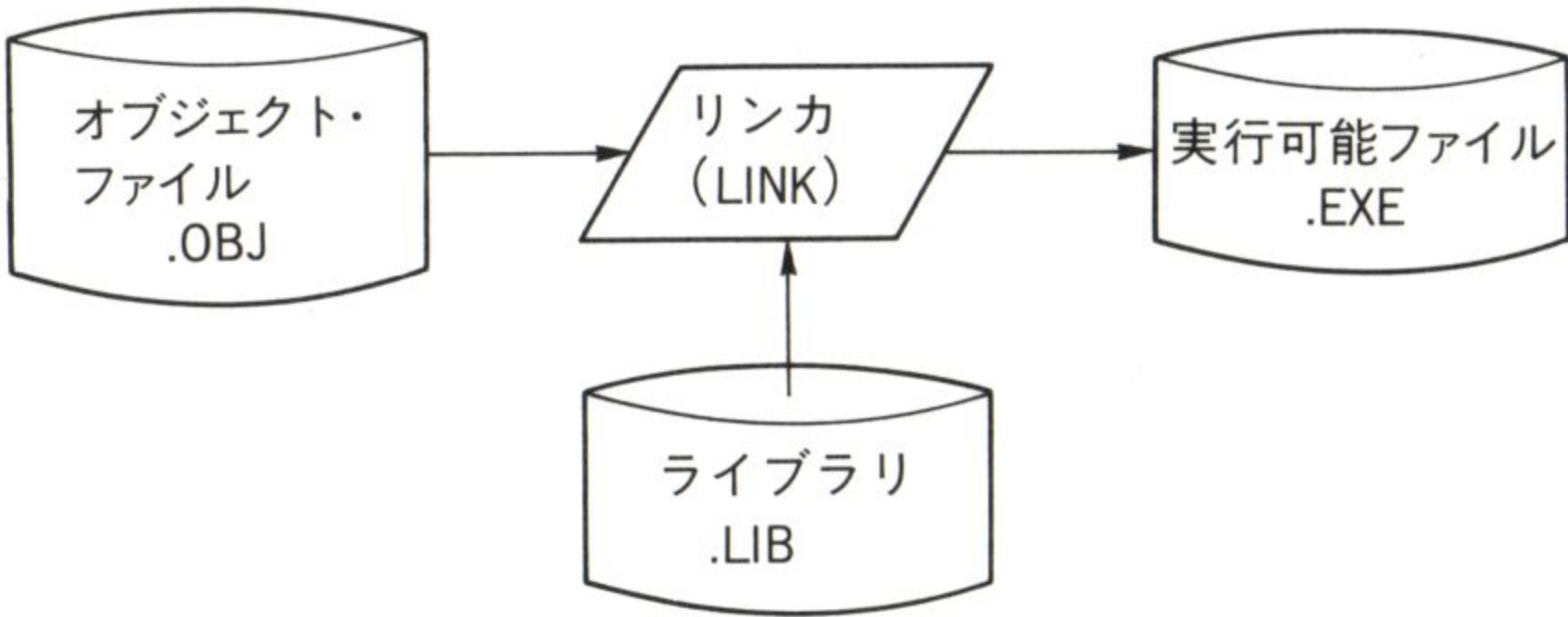
■/ZD /ZI (デバッガ用コードの指定)

/ZD は，SYMDEB でデバッグするためのデバッグ情報を埋め込みます．

/ZI は，Code View でデバッグするためのデバッグ情報を埋め込みます．

3-2 リンカ(LINK)

リンカは，リロケータブル・オブジェクト・ファイル(.OBJ) から実行可能ファイル(.EXE)を生成します．その際必要に応じてライブラリを結合します．

オブジェクト・ファイル .OBJ → リンカ (LINK) → 実行可能ファイル .EXE
ライブラリ .LIB → リンカ (LINK)

1 LINK の起動法

LINK はデフォルト・ライブラリおよび Libraries プロンプトに対しファイル名のみ与えられたライブラリについては，次の順序でライブラリ・ファイルを探し，ユーザ・オブジェクト・ファイルに結合します．

1　カレント・ディレクトリ
2　Libraries [.LIB]プロンプトに対して指定したディレクトリ
3　LIB 環境変数で指定されているディレクトリ

パスの指定されたライブラリ・ファイルは，上の順序とは関係なく指定されたパス位置を探します．

Quick BASIC の標準ライブラリ(BCOM42A.LIB，BCOM42E.LIB，BRUN42A.LIB，BRUN42E.LIB)は，コンパイル時に，どのライブラリを結合するかの指示がオブジェクト・ファイル中に埋め込まれているので，Libraries プロンプトに対して指定する必要はありません．

LINK に入力する4つのファイルの意味は次のとおりです.

▼ LINK に入力するファイル

ファイル	機　能
オブジェクト・ファイル Object Modules[.OBJ]	リンクするオブジェクト・モジュール. オブジェクトが複数ある場合は, プラス(+)または空白(␣)で区切る. リンカは並べられた順に同じクラス名のセグメントをまとめてロードする
ラン・ファイル Run File[Object.EXE]	実行可能ファイル. ファイル・タイプには.EXE 以外を与えない方がよい. ファイル名を省略するとオブジェクト・ファイル・リストの先頭のファイル名が採用される
リスト・ファイル List File[NUL. MAP]	ロードされたモジュールの各セグメントに対するエントリをマップしたリスト・ファイル
ライブラリ・ファイル Libraries[.LIB]	ライブラリ・ファイルはライブラリ・マネジャ LIB により作成されたものでなければならない. ライブラリが複数ある場合は, プラス(+)または空白(␣)で区切る

LINK の起動法には次の3つの方法があります.

■方法1　単独で起動する

```
A>LINK↵

Object Modules[.OBJ]: オブジェクト・ファイル・リスト名↵
Run File[Object. EXE ]: ラン・ファイル名↵
List File [NUL. MAP]: リスト・ファイル名↵
Libraries [.LiB]: ライブラリ・ファイル・リスト名↵
```

各プロンプトに対応するファイル名を入力します. ↵のみ入力すると, [　]で囲まれたデフォルトのファイル名が付けられます. ファイルのタイプを省略すると, [　]内のタイプが採用されます. 各応答の最後に"/"で始まるスイッチを指定することができます.

プロンプトに対してセミコロン(;)を入力すると, それ以後のプロンプトは表示されず, デフォルトのファイル名が採用されます.

NUL とはファイルを何も作らないということを意味しています. ファイルはカレント・ドライブのカレント・ディレクトリに作られますので, ほかへ作りたいときはドライブ名またはパス名を付けてください.

〈オブジェクト・ファイル・リスト名〉と〈ライブラリ・ファイル・リスト名〉に書く各ファイルは空白またはプラス(+)で区切ります. 1行に書ききれない場合は, 行の最後に+を置いて↵を入力すると次の行に継続できます.

例

```
A>LINK ⏎
Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: test ⏎
Run File [TEST.EXE]: ⏎
List file [NUL.MAP]: test ⏎
Libraries [.LIB]: ⏎
```

■方法2　ファイル名をコマンド・ラインに与えて起動する

A>LINK　[／オプション]オブジェクト・リスト，[ラン・ファイル]，[リスト・ファイル]，[ライブラリ・リスト]⏎

ファイルは方法1の場合と同様のものです．ファイル名を指定せずにカンマ(,)のみを置くと，そのファイル名はデフォルト解釈されます．また，セミコロン(；)を置くと，それ以後のファイル名はすべてデフォルト解釈されます．

例

A>LINK B：TEST, B：,, MYLIB⏎

…ドライブBの"TEST.OBJ"をロードし，MYLIB.LIBとリンクして"TEST.EXE"という実行可能ファイルをドライブBに作成する．

A>LINK TEST＋SMOD1＋SMOD2；⏎

…"TEST.OBJ", "SMOD1.OBJ", "SMOD2.OBJ"をリンクして，"TEST.EXE"を作る．

■方法3　応答ファイル名をコマンド・ラインに与えて起動する

A>LINK　@filename⏎

filenameで示される応答ファイルには，LINKに応答する4種類のファイル名をあらかじめエディタで作成しておきます．filenameのファイル・タイプは何であってもかまいません．

たとえば，次のような応答ファイルLINKFをエディタで作成したとします．

▼応答ファイル LINKF

```
B：TEST⏎
B：⏎
⏎
MYLIB⏎
```

このとき，

```
A>LINK  @LINKF ↵
```

とすると，“B:TEST.OBJ”が“MYLIB.LIB”とリンクされ，“B:TEST.EXE”ファイルが作成されます．

2 LINKのリンカ・オプション

リンカ・オプションはLINKに対し細かな指示をするためのもので，スラッシュ(/)で始まる英数文字列です．オプション文字の大/小は区別されません．

リンカ・オプションは，フル・スペルと省略形の2通りの指定が可能で，一般に省略形を用います．たとえば，/BATCHは/Bと省略して書けます．

以下にQuick BASICのプログラムをリンクする際によく使う，リンカ・オプションを説明します．

■/B　(BATCH：リンカの継続)

リンカが，ライブラリやオブジェクト・ファイルを見つけられなくても，そのままリンカ処理を続けさせます．

■/CO　(CODEVIEW：デバッガ用オブジェクトの生成)

Code Viewデバッガ用のオブジェクトを生成します．

コンパイラ・オプション/ZIを指定してコンパイルしたオブジェクトに対して有効です．

■/E　(EXEPACK：実行可能ファイルのパック)

作成するEXEファイル中の初期化データの中で，同一の文字が並んでいるものを圧縮し，リロケーション・テーブルを最適化します．これによりファイル・サイズの縮小とロード時間の短いEXEファイルが生成されます．

■/HE　(HELP：ヘルプ表示)

利用できるリンカ・オプションの一覧を表示します．

■/INF　(INFOMATION：リンク情報の表示)

リンク情報(リンクのフェーズやリンク中のオブジェクト・ファイル名など)を表示します．

■/LI　(LINENUMBERS：行番号付きのリスティング)

ソース・ファイルに対応した行番号付きのリスティング・ファイルを作成します．

■/M[：n]　(MAP：マップ・ファイルの作成)

マップ・ファイルを作成します．nはマップ・リスト中のソートされるシンボルの数で1～65535です．デフォルトは2048です．

■/NOD　(NODEFAULTLIBRARYSEARCH：デフォルト・ライブラリの無視)

オブジェクト・ファイルに埋め込まれているデフォルトの標準ライブラリの結合指示を無視します．

このオプションを指定した場合は，Libraries プロンプトに対し，次の順序でライブラリを指示します．

1．ユーザライブラリ
2．Quick BASIC の標準ライブラリ

■/NOE　(NOEXTDICTIONARY：拡張ディクショナリを検索しない)

拡張ディクショナリ(リンカが保守するシンボル位置の内部リスト)を検索しません．このオプションはライブラリ・シンボルが再定義されるときに使います．

■/NOI　(NOIGNORECASE：大/小文字の区別)

大文字と小文字を区別します．つまりグローバル・シンボル upper と UPPER は異なるものと判断します．

■/NOP　(NOPACKCODE：セグメントのグループ化をしない)
/PAC[：n]　(PACKCODE：セグメントのグループ化をする)

連続する論理コード・セグメントをグループ化する(/PAC)，しない(/NOP)を指定します．

nは新しいグループ化を始める範囲で，デフォルトは64K です．

■/PAU　(PAUSE：リンクの一時停止)

実行可能ファイルをディスクに書き出す前にメッセージを出して，リンク作業を一時停止します．これにより，実行可能ファイルを別のディスクにセーブすることができます．[↵]を押すとリンクが再開されます．

■/Q　(QUICKLIB：クイック・ライブラリの作成)

オブジェクト(.OBJ) を実行可能ファイル(.EXE)にせずに，クイック・ライブラリ(.QLB)として作成します．

/Qオプションを使用する場合は，クイック・ライブラリ・サポートルーチンBQLB42.LIB をライブラリ・リストに必ず指定してください．

例

```
LINK  /Q GCOPY, GCOPY.QLB,,BQLB42.LIB
```

…GCOPY.OBJ からクイック・ライブラリ GCOPY.QLB を作成

■/SE:n　(SEGMENTS：セグメント数の指定)

プログラムに許されるセグメント数をn(1～1024)に制限します．デフォルトは128です．

3−3 ライブラリ・マネジャ(LIB)

1 ライブラリ・マネジャとは

利用価値の高いルーチンを共通の資源として使用できるようにすることは，プログラム開発の生産性を向上させます．

リロケータブル・オブジェクト(.OBJ) のレベルでLINK時にリンクすることも考えられますが，これが何本にも増えてくるとひとかたまりのライブラリ・ファイルにしておいたほうが便利です．リロケータブル・オブジェクト(.OBJ)をライブラリ・ファイル(.LIB)にするのがライブラリ・マネジャ(LIB) です．

Quick BASIC では，

- **スタンドアロン・ライブラリ(.LIB)**
- **クイック・ライブラリ(.QLB)**

という2種類のライブラリを扱うことができます．

ライブラリ・マネジャ LIB は，スタンドアロン・ライブラリを管理するものです．クイック・ライブラリについては第2章2-6の3を参照してください．

さて，ライブラリ・マネジャについて説明する前に，オブジェクト・ファイルとオブジェクト・モジュールという語についてここで定義しておくことにします．

オブジェクト・ファイルとは，コンパイル(またはアセンブル)により生成されたリロケータブル・オブジェクトです．たとえば，

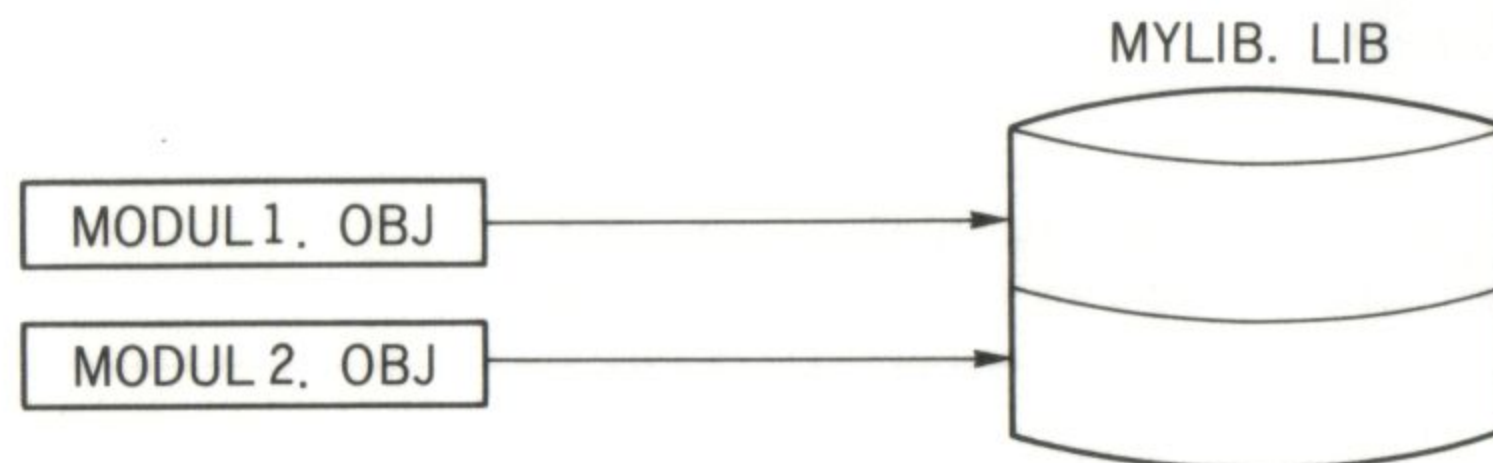
```
B:MODUL1.OBJ
```

のようにドライブ名とファイル・タイプ(.OBJ)を持つものです．

一方，オブジェクト・モジュールとは，オブジェクト・ファイルをLIB(ライブラリ・マネジャ) によりライブラリとしたときの個々のモジュールを指します．たとえば，上の例のドライブ名とファイル・タイプをとったMODUL1がライブラリ・ファイル内での名前となります．

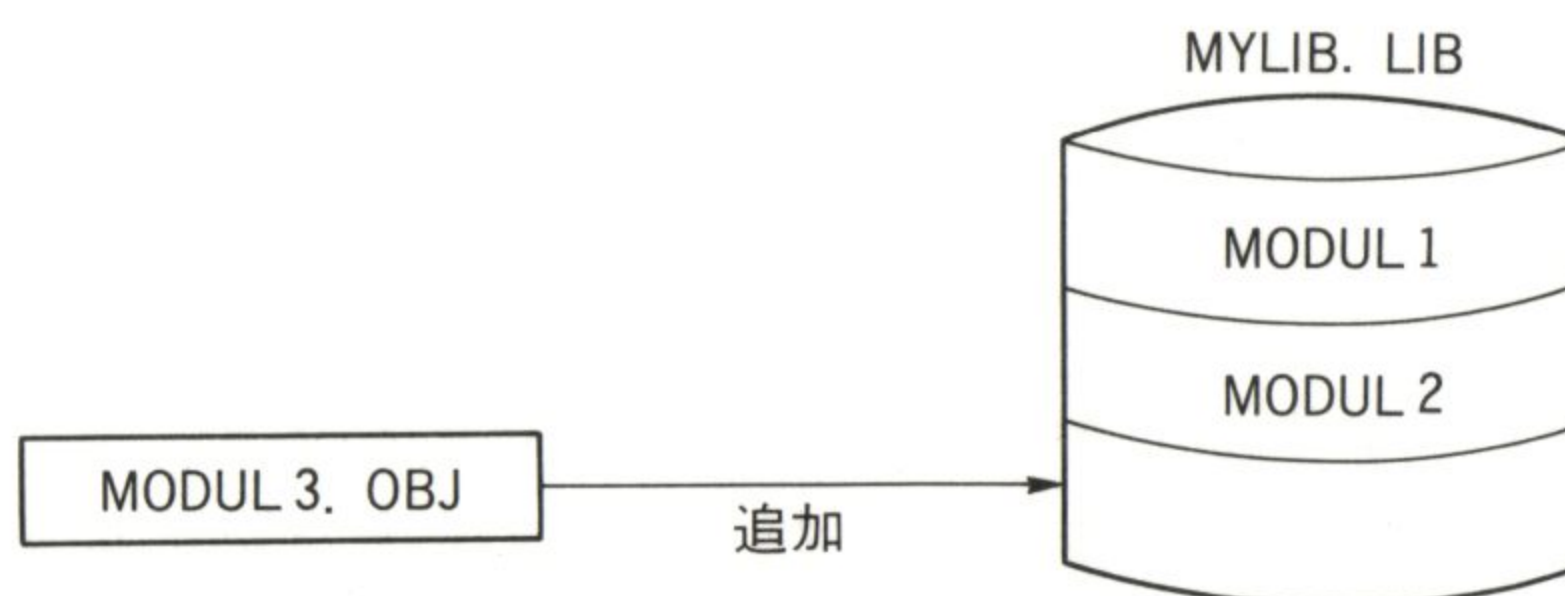
ライブラリ・マネジャの機能の概要を以下に紹介します．

■ライブラリの新規作成

一般に，ユーザ・ライブラリは何もない状態から新規に作成されていきます．そこで，MODUL1.OBJ と MODUL2.OBJ というリロケータブル・オブジェクトを MYLIB.LIB として登録します．

MYLIB. LIB
MODUL1. OBJ
MODUL 2. OBJ

```
A>lib ↵

Microsoft (R) Library Manager  Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib ↵
Library does not exist.  Create? (y/n) y ↵
Operations:+modul1+modul2 ↵
List file: ↵
```

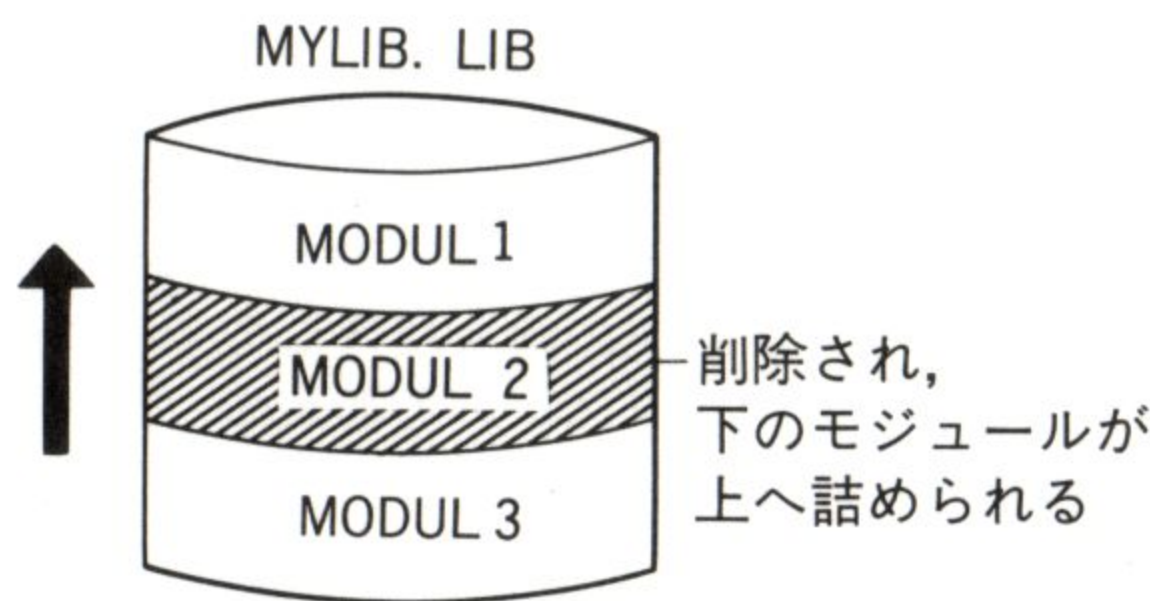
■ライブラリへの追加

新規作成されたライブラリ(MYLIB.LIB)へ新しいモジュール(MODUL3.OBJ)を追加します．基本的にはライブラリへの追加も，ライブラリの新規作成も同じ操作です．唯一違うのは，新規作成においてはライブラリ・ファイルが存在しないので，LIB からの問い合わせに対して“Y”と答えることだけです．

MYLIB. LIB
MODUL 1
MODUL 2
MODUL 3. OBJ
追加

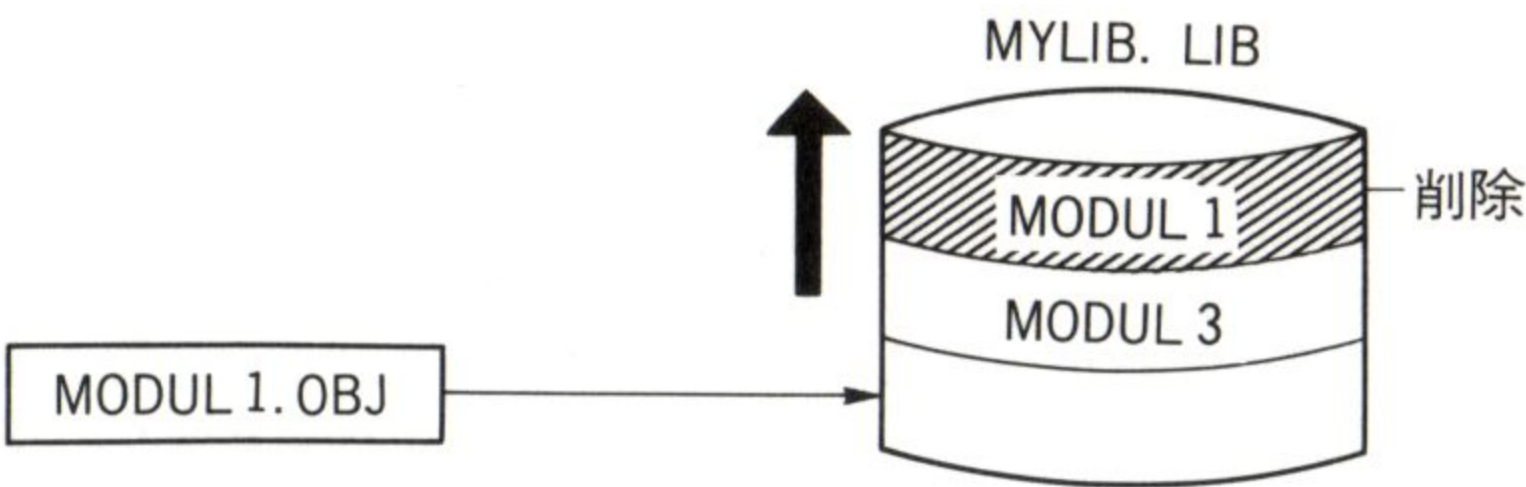
```
A>lib ↵

Microsoft (R) Library Manager  Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib ↵
Operations:+modul3 ↵
List file: ↵
Output library:mylib ↵
```

■ライブラリからの削除

ライブラリ(MYLIB.LIB)から不要になったモジュール(MODUL2)を削除します．MODUL2のあった位置には，下位にあったMODUL3が詰められるので，削除を繰り返してもライブラリ内には空きが生じないようになっています．

MYLIB. LIB
MODUL 1
MODUL 2
MODUL 3
削除され，
下のモジュールが
上へ詰められる

```
A>lib⏎

Microsoft (R) Library Manger   Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib ⏎
Operations:-modul2 ⏎
List file: ⏎
Output library:mylib ⏎
```

■ライブラリのモジュール置換

ライブラリ MYLIB.LIB の MODUL1を，新しいオブジェクト・ファイル MODUL1.OBJ と置換します．実際には削除と追加が行われます．

MYLIB. LIB
MODUL 1
削除
MODUL 3
MODUL 1. OBJ

```
A>lib ⏎

Microsoft (R) Library Manager  Version 3.11
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Library name:mylib ⏎
Operations:-+modul1 ⏎
List file: ⏎
Output library:mylib ⏎
```

2 LIB の起動法

LIB に入力する 4 つのファイルの意味は以下のとおりです．

▼ LIB に入力するファイル

ファイル	意味
ライブラリ・ファイル Library name：	ライブラリのファイル名．デフォルトのファイル・タイプは.LIB ライブラリが新規の場合だけこの応答に対し，LIB は問い合わせを出すので"Y"または"y"と答える．それ以外の文字を入力すると処理は中断する
オペレーション Operations：	コマンド・キャラクタとそれに続くモジュール名を並べたもので，LIB に対する処理内容の指示となる．モジュール名のデフォルト・タイプは.OBJ．このプロンプトに⏎のみを入力すると，ライブラリの整合性のチェックのみを行う
リスト・ファイル List file：	ライブラリの中にあるモジュールの PUBLIC シンボルのクロス・リファレンス・リスティングを収めるファイル
出力ライブラリ・ファイル Output library：	現在のライブラリ・ファイル名と異なるライブラリ・ファイル名を指定できる．デフォルト(⏎)は現在のライブラリ・ファイル名でライブラリが作成される．このとき前のライブラリは.BAK というバックアップ・ファイルに変更される

オペレーションに書くコマンド・キャラクタには次のようなものがあります．－＋と－＊は，Ver.3.0における追加キャラクタです．

▼コマンド・キャラクタ

キャラクタ	機能
＋	ライブラリの最後部にオブジェクト・ファイルを追加
－	ライブラリの 1 つのモジュールを削除
＊	ライブラリからモジュールを切り出しオブジェクト・ファイルを作成
－＋	ライブラリのモジュールを同名のオブジェクト・ファイルで置き換え
－＊	指定したオブジェクト・モジュールをライブラリから抜き取り，オブジェクト・ファイルに収める
;	残りのプロンプトに対し，デフォルトの応答をする
&	コマンド行に書ききれなくなったときに次の行へ継続

LIB の起動法には次の3つの方法があります.

■方法1　単独で起動する

A>LIB↵
Library name: ライブラリ・ファイル名[/PAGESIZE:n]↵
Operations: オペレーション↵
List file [NUL.MAP]: リスト・ファイル名↵
Output library: 出力ライブラリ・ファイル名↵

各プロンプトに対応するファイル名またはオペレーションを入力します. ↵のみ入力すると,[　]で囲まれたデフォルトのファイル名が付けられます. プロンプトに対してセミコロン(;)を入力すると, それ以後のプロンプトは表示されず, デフォルトのファイル名が採用されます.

■方法2　ファイル名をコマンド・ラインに与えて起動する

A>LIB ライブラリ・ファイル名[/PAGESIZE:n][オペレーション]
[,リスト・ファイル][,出力ライブラリ・ファイル名]↵

ライブラリ・ファイル名とオペレーションの区切りは, コマンド・キャラクタで分離します. コマンド・ライン の各項目の途中にセミコロン(;)を置くと, それ以後の項目はデフォルト値が採用されます.

例

A>LIB MYLIB −+DISP ; ↵
…ライブラリ MYLIB のモジュール DISP を, 新しいオブジェクト・ファイル DISP.OBJ で更新.

A>LIB MYLIB ; ↵
…ライブラリ MYLIB の整合性のチェックのみを行う.

A>LIB MYLIB +B : DISP−MODUL1 ; ↵
…ライブラリ MYLIB にドライブ B の DISP.OBJ を追加し, MODUL1 を削除.

A>LIB MYLIB +MODUL1+MODUL2+MODUL3, MYLIB.LST↵
…MODUL1.OBJ, MODUL2.OBJ, MODUL3.OBJ をライブラリ MYLIB.LIB に登録. このとき MYLIB.LST というリスト・ファイルを生成する.

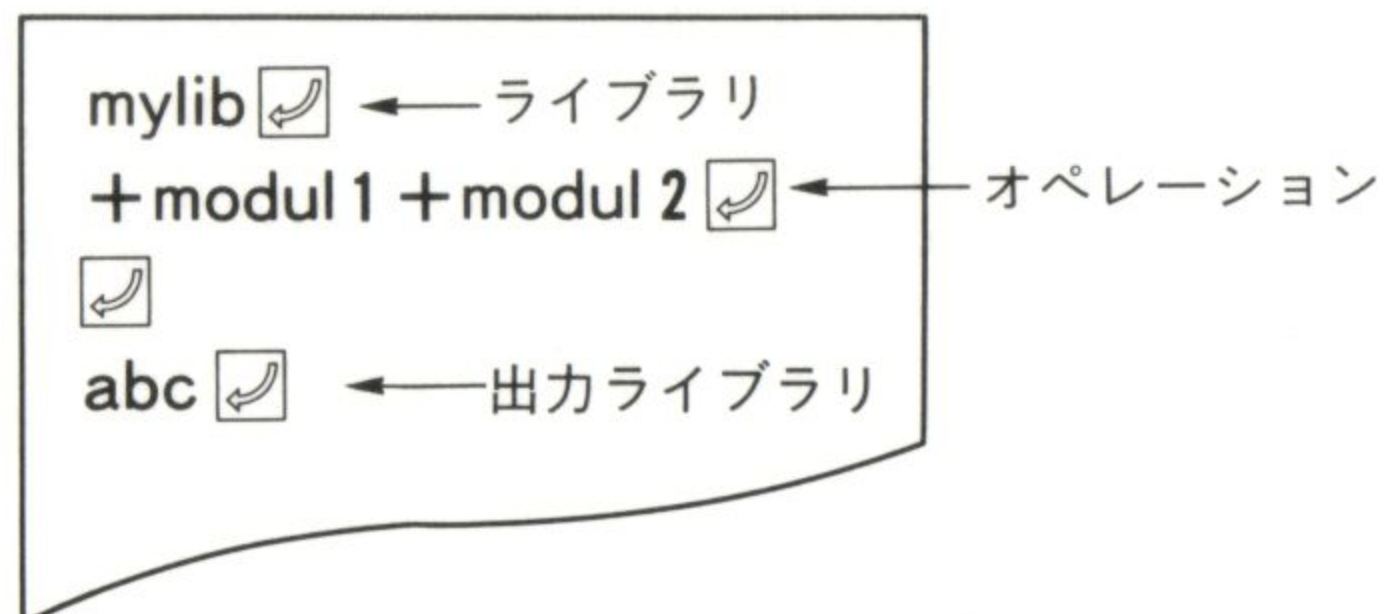
▩方法 3　応答ファイル名をコマンド・ラインに与えて起動する

```
A>LIB @filename↵
```

filenameで示される応答ファイルには，LIB に応答する 4 種類のファイル名(またはオペレーション)をあらかじめエディタで作成しておきます．filenameのファイル・タイプは何であってもかまいません．

filenameの例を次に示します．

```
mylib↵                      ←ライブラリ
+modul 1 +modul 2↵          ←オペレーション
↵
abc↵                        ←出力ライブラリ
```

第4章

Quick BASICの言語仕様

4－1 プログラムの基本構成要素

1 Quick BASICの文字セット

Quick BASIC のプログラムで使用できる文字は，

- **アルファベットの大/小文字**
- **数字**
- **特殊文字**

です．特殊文字はプログラム中で特別な意味を持つもので，以下のものです．

特殊文字	名　称	機　能
	改行コード	行の終了
␣	スペース：空白	文要素の区切り
!	エクスクラメイション：感嘆符	単精度データ型を表す
#	シャープ：ナンバー	倍精度データ型を表す
$	ドル	文字列データ型を表す
%	パーセント	整数データ型を表す
&	アンパサンド	倍長整数データ型を表す
'	シングル・クォーテーション：単一引用符	コメントの開始
(	レフト・パレンシシス：左かっこ	式要素の始め，配列添字の始め
)	ライト・パレンシシス：右かっこ	式要素の終わり，配列添字の終わり
*	アスタリスク	乗算演算子，固定長文字列の長さ指定
+	プラス・サイン：正符号	加算演算子
,	カンマ：句読点	リストの要素の区切り
−	マイナス・サイン：負符号	減算演算子，範囲
.	ピリオド：終止符	小数点，レコードメンバの参照
/	スラッシュ	除算演算子
:	コロン	ステートメントの区切り
;	セミコロン	リストの要素の区切り
<	レフト・アングル・ブラケット：左山かっこ	比較演算子
=	イコール：等号	代入記号，比較演算子
>	ライト・アングル・ブラケット：右山かっこ	比較演算子
?	クェスション：疑問符	PRINT の略字
@	アット・マーク	可変長文字列フィールド
[	レフト・ブラケット：左大かっこ	配列添字の始め
]	ライト・ブラケット：右大かっこ	配列添字の終わり
¥	円記号	整数除算演算子
^	カレット：脱字記号	べき乗演算子
_	アンダー・スコア：下線	プログラム行の継続

なお，コメント中には，

- **半角カタカナ**
- **2バイト文字(仮名，漢字)**

を使用することができます．

2 プログラム行の構成

プログラムの各行は次の構成となります．

[行識別子]　ステートメント[：ステートメント][’コメント]

行の終わりは↵で示し，1行に最大256文字まで書けます．

1行に複数のステートメント(命令文)を書くことができます．この場合は，各ステートメントをコロン(：)で区切ります．このように，1行に複数のステートメントをコロン(：)で区切って書くことを，マルチ・ステートメントといいます．

●行識別子

QBでは従来型BASICのような行番号は不要です．ただしGOTO文の飛び先として行の先頭に行識別子を置くことができます．

行識別子としては行番号(100，200など)とラベル(OWARI：など)が使用できます．

ラベル名の付け方のきまりは変数名の付け方のきまりと同じです．

●ステートメント(命令文)

ステートメントはプログラムを構成する単一の仕事を記述したもので，命令語とオペランドとからなります．命令語とは，IF，FOR，PRINTなどのBASICがあらかじめ定めている言葉です．オペランドには，定数，変数，関数，そしてこれらを演算子で結んだ式を指定します．

```
PRINT     A+B
命令語    オペランド
  ステートメント(命令文)
```

●コメント

コメントは単一引用符(’)に続いて書きます．コメントの開始に単一引用符(’)を用いた場合はコロン(：)で区切る必要はありませんが，REMを用いた場合はコロン(：)で区切らなければなりません．

```
Num=40    ’生徒の人数
Num=40    ：REM 生徒の人数
```

注）

QB のエディタ以外のエディタを使ってプログラムを作成した場合，1 行に収まらない場合に，行末に下線(_)を置くことで，次の行に続けることができます．このプログラムを QB 上にロードすると，下線(_)は削除され，続く行が結合されて 1 つの行になります．
ただし，QB のエディタ上で，下線(_)を使用して行を継続することはできません．

3 予約語

IF，FOR，PRINT などの命令語と，SIN，COS などの組み込み関数の関数名は，あらかじめ BASIC が定めている言葉で，これを予約語と呼びます．

Quick BASIC の予約語は以下のとおりです．

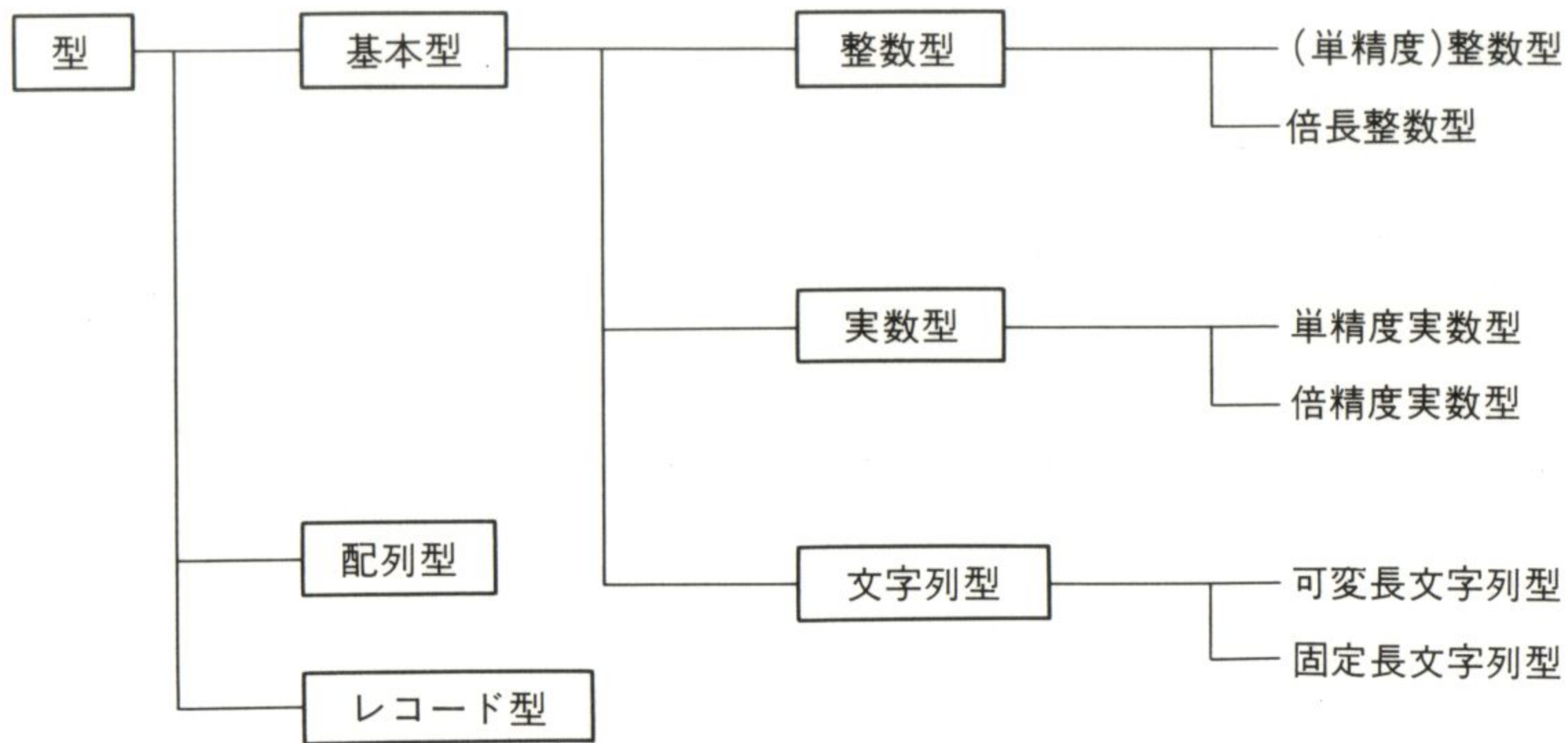
ABS	CSNG	EXIT	LOC
ACCESS	CSRLIN	EXP	LOCAL
ALIAS	CVD	FIELD	LOCATE
AND	CVDMBF	FILEATTR	LOCK
ANY	CVI	FILES	LOF
APPEND	CVL	FIX	LOG
AS	CVS	FOR	LONG
ASC	CVSMBF	FRE	LOOP
ATN	DATA	FREEFILE	LPOS
BASE	DATE$	FUNCTION	LPRINT
BEEP	DECLARE	GET	LSET
BINARY	DEF	GOSUB	LTRIM$
BLOAD	DEFDBL	GOTO	MID$
BSAVE	DEFINT	HEX$	MKD$
BYVAL	DEFLNG	IF	MKDIR
CALL	DEFSNG	IMP	MKDMBF$
CALLS	DEFSTR	INKEY$	MKI$
CASE	DIM	INP	MKL$
CDBL	DO	INPUT	MKS$
CDECL	DOUBLE	INPUT$	MKSMBF$
CHAIN	DRAW	INSTR	MOD
CHDIR	ELSE	INT	NAME
CHR$	ELSEIF	INTEGER	NEXT
CINT	END	IOCTL	NOT
CIRCLE	ENDIF	IOCTL$	OCT$
CLEAR	ENVIRON	IS	OFF
CLNG	ENVIRON$	KEY	ON
CLOSE	EOF	KILL	OPEN
CLS	EQV	LBOUND	OPTION
COLOR	ERASE	LCASE$	OR
COM	ERDEV	LEFT$	OUT
COMMAND$	ERDEV$	LEN	OUTPUT
COMMON	ERL	LET	PAINT
CONST	ERR	LINE	PALETTE
COS	ERROR	LIST	PCOPY

PEEK	RIGHT$	SPACE$	TROFF
PEN	RMDIR	SPC	TRON
PLAY	RND	SQR	TYPE
PMAP	RSET	STATIC	UBOUND
POINT	RTRIM$	STEP	UCASE$
POKE	RUN	STICK	UNLOCK
POS	SADD	STOP	UNTIL
PRESET	SCREEN	STR$	USING
PRINT	SEEK	STRIG	VAL
PSET	SEG	STRING	VARPTR
PUT	SELECT	STRING$	VARPTR$
RANDOM	SETMEM	SUB	VARSEG
RANDOMIZE	SGN	SWAP	VIEW
READ	SHARED	SYSTEM	WAIT
REDIM	SHELL	TAB	WEND
REM	SIGNAL	TAN	WHILE
RESET	SIN	THEN	WIDTH
RESTORE	SINGLE	TIME$	WINDOW
RESUME	SLEEP	TIMER	WRITE
RETURN	SOUND	TO	XOR

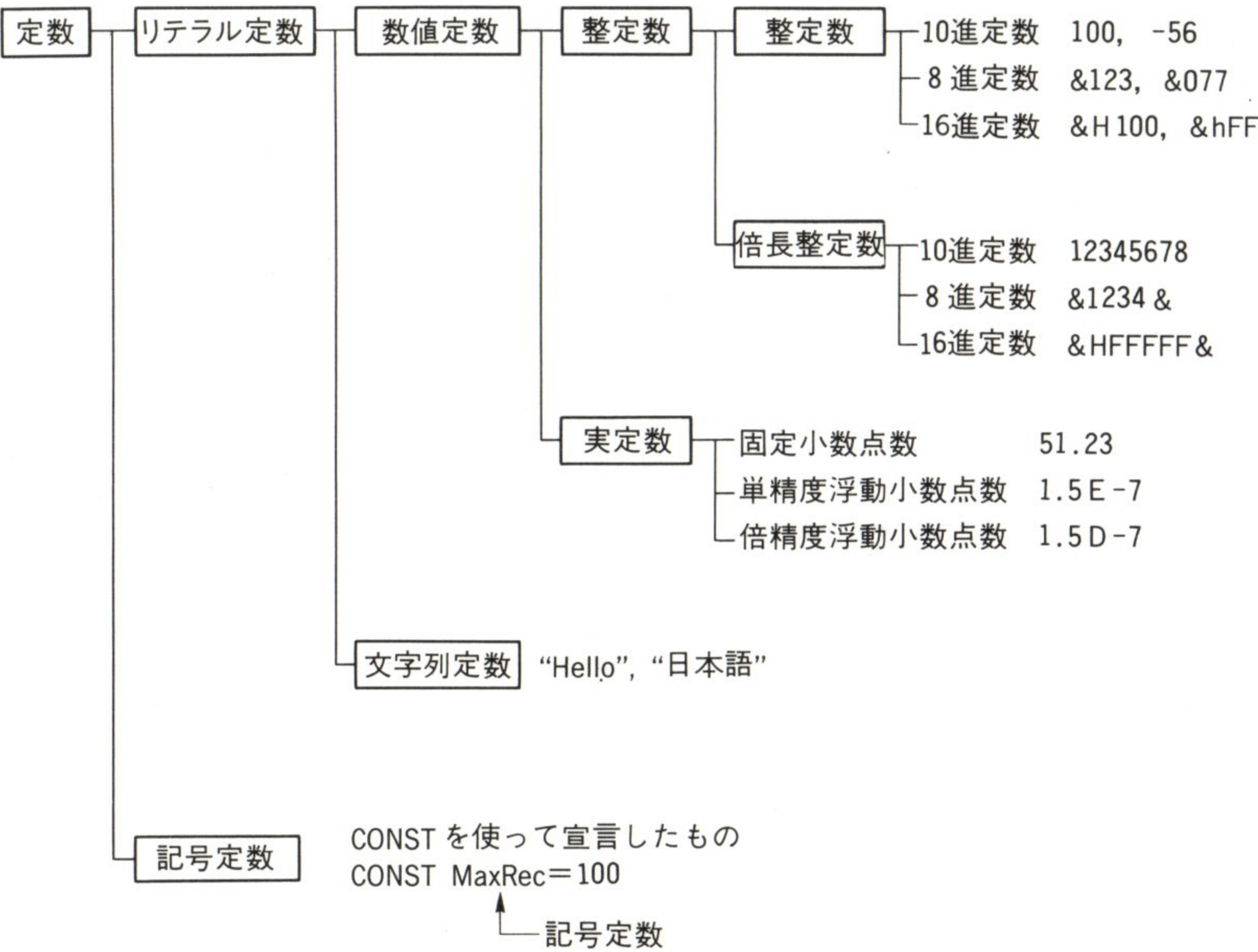
4−2 データ型

1 データ型の種類

Quick BASICには次のようなデータ型があります.

- 型
  - 基本型
    - 整数型
      - (単精度)整数型
      - 倍長整数型
    - 実数型
      - 単精度実数型
      - 倍精度実数型
    - 文字列型
      - 可変長文字列型
      - 固定長文字列型
  - 配列型
  - レコード型

2 定数

プログラムの実行中に変化しない値を定数といい，以下のように分類されます．

- 定数
  - リテラル定数
    - 数値定数
      - 整定数
        - 整定数
          - 10進定数　100，-56
          - 8進定数　&123，&077
          - 16進定数　&H100，&hFF
        - 倍長整定数
          - 10進定数　12345678
          - 8進定数　&1234&
          - 16進定数　&HFFFFFF&
      - 実定数
        - 固定小数点数　51.23
        - 単精度浮動小数点数　1.5E-7
        - 倍精度浮動小数点数　1.5D-7
    - 文字列定数　"Hello"，"日本語"
  - 記号定数　CONSTを使って宣言したもの
    CONST MaxRec＝100（MaxRec：記号定数）

■リテラル定数

literal は「文字どおりの」という意味です．リテラル定数とは，100や"Hello"のような数値または文字列そのものを表す定数です．

リテラル定数は数値定数と文字列定数に別れます．数値定数の進数および精度は以下のような記号を用いて示します．

- 8進数を示すには，先頭に&O，&oまたは&を付けます．
- 16進数を示すには，先頭に&Hまたは&hを付けます．
- 8進または16進の倍長整定数を示すには，末尾に&を付けます．
- 単精度浮動小数点数の指数を示す記号としてEまたはeを用います．
- 倍精度浮動小数点数の指数を示す記号としてDまたはdを用います．

■記号定数

Quick BASIC ではCONST 文により記号定数を宣言することができます．記号定数とは，定数に名前を割り当て，その名前を定数として扱えるようにしたものです．

記号定数は次のように定義します．

CONST　記号定数名＝値，記号定数名＝値…

記号定数のデータ型は，名前にその型を宣言する文字を付けなければ，割り当てられる値(式)のデータ型になります．

記号定数名の付け方のきまりは変数名の付け方のきまりと同じです．

例

```
CONST  Boy=20, Girl=21, Seito=Boy+Girl
```
… Seito は 20+21=41 と定義される．
```
CONST  Pai#=3.141592657#
```
…記号定数の末尾に付いた%，&，!，#，$は定数名の一部ではなくデータ型を示す．

3 変数

■名前のきまり

プログラムの中で使う値を格納するための箱(実際はメモリ)を変数といい，その箱に付ける名前を変数名といいます．

変数名は次の規則で付けます．

- 変数名の先頭は英字で始まります．
- 第2文字以後は英字，数字，ピリオド(.)が指定できます．
- 変数名の長さは最大40文字です．
- 予約語は使用できませんがそれを含んだものはかまいません．
- FN で始まる変数名は許されません．
- 英大文字と英小文字は同じものとして扱われます．つまり，SUM，Sum，sum などはすべて同じものとして扱われます．

記号定数，プロシージャ，関数，ラベルに付ける名前も，この規則に従います．

■変数の型宣言

Cや Pascal ではプログラムで使用する変数は，必ず型宣言しなければなりませんが，BASIC はこうした制約はなく，変数の型宣言をせずに使用した変数は単精度実数型とみなされます．これを「暗黙の型宣言」といいます．

このような暗黙の型宣言によらず変数の型を明示的に宣言するには，次のような3つの方法があります．

●変数名の後に型宣言文字を付加する

A$，A%のように変数名の後に型宣言文字を付加して，その変数の型を宣言することができます．型宣言文字として以下のものがあります．

%	(単精度)整数型
&	倍長整数型
!	単精度実数型
#	倍精度実数型
$	文字列型

注）A%，A&，A!，A #，A$はすべて異なる変数です．

●宣言文を使って宣言する

次のように宣言子と型名を指定して変数の型を宣言することができます．

宣言子　変数名,変数名,…AS　型名

宣言子:

COMMON	… 共用
DIM	… 配列
REDIM	… 動的配列の再定義
SHARED	… プロシージャ間の共用
STATIC	… ローカル変数

型名:

INTEGER	… (単精度)整数型
LONG	… 倍長整数型
SINGLE	… 単精度実数型
DOUBLE	… 倍精度実数型
STRING	… 可変長文字列型
STRING ＊長さ	… 固定長文字列型
タグ名	… ユーザ定義型(レコード型)

例

```
DIM A AS STRING *10
```

…変数 A を長さ10文字の固定長文字列型変数として宣言

```
DIM Sum AS INTEGER
```

…変数 Sum を整数型変数として宣言

●インプリシット宣言

DEF typ 文により，変数の先頭文字で型を分類させることができます．

```
{DEFINT }
{DEFLNG }
{DEFSNG } アルファベット[-アルファベット],…
{DEFDBL }
{DEFSTR }
```

例

DEFINT I-N
　　…I～N で始まる変数(Jac, Man など) を整数型として扱うことを宣言

DEFDBL A, B, X-Z
　　…A, B, X～Z で始まる変数を倍精度実数型として扱うことを宣言

DEFINT A-Z
　　…すべての変数を整数型として扱うことを宣言

●宣言の優先順位

型宣言文字を付加した宣言と，宣言文を使った宣言は，DEFtyp 文に対し常に優先します．

たとえば，

```
DEFINT   I-N
DIM  Num  AS  LONG
```

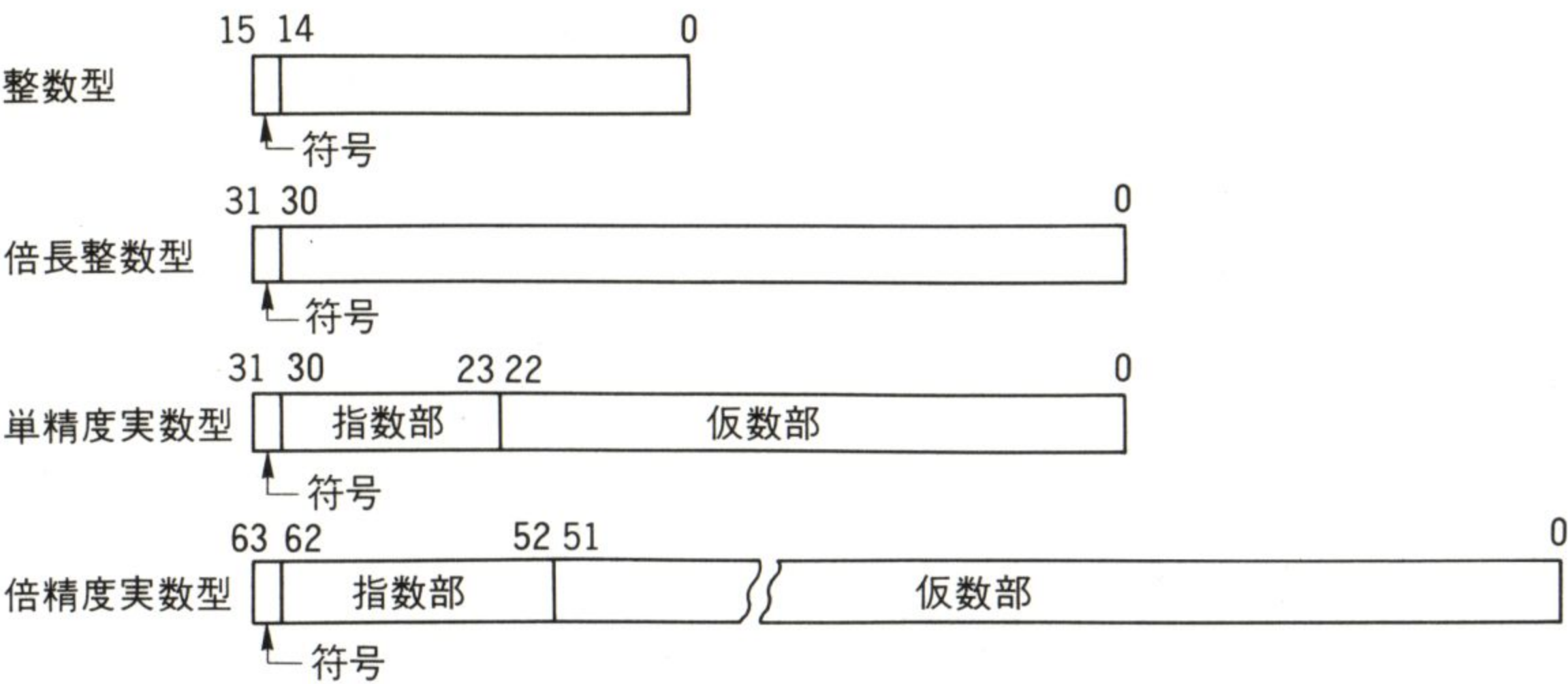
のような宣言において，Num は倍長整数型として宣言されたことになります．

4 基本データ型

Quick BASIC において，あらかじめ定められた型(いわゆる基本データ型)のサイズとデータ範囲を以下に示します．

データ型	サイズ	扱える値の範囲	説　　明
整数型	2バイト	−32768〜32767	100や5という整数を扱う型
倍長整数型	4バイト	−2147483648 〜2147483647	100000のような有効桁の長い整数を扱う型
単精度実数型	4バイト	−3.402823E＋38 〜3.402823E＋38	3.14159のような小数点数を扱う型．7桁
倍精度実数型	8バイト	−1.79769313486231D+308 〜1.79769313486231D+308	3.1415927…のような有効桁の長い小数点数を扱う型.16桁
文字型	可変長	最大32767文字	"Hello"のような文字を扱う型
	固定長		

注）E＋38は10^{38}を意味します．
　　D＋308は10^{308}を意味します(D は Double の意味)．

整数型，倍長整数型，単精度実数型，倍精度実数型の内部データ形式を以下に示します．

■固定長文字列

一般に BASIC の文字列は可変長文字列で，文字列変数に入れる文字列の長さは自由自在です．Quick BASIC ではこのような可変長文字列のほかに，固定長文字列をサポートしています．

固定長文字列の宣言は，型指定子 STRING を用いて次のように行います．

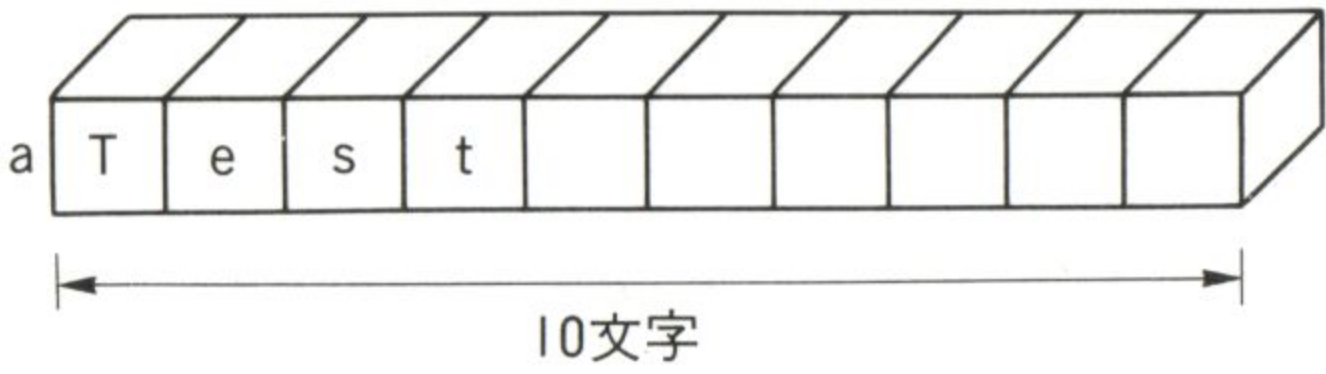
```
DIM　変数名　AS　STRING ＊長さ
```

たとえば，

```
DIM　a　AS　STRING ＊10
```

と宣言すると，変数 a は10文字の文字を入れる固定長文字列変数となります．

```
a ＝ "Test"
```

とすると，文字は次のように左づめで格納され，残った右は空白で埋められます．

したがって，

```
IF a＝ "Test" THEN
```

としても，この条件は一致しません．これは，

```
IF a＝ "Test␣␣␣␣␣␣" THEN
```

としなければなりません．

後述するレコード型のメンバには可変長文字列を指定できませんので，この固定長文字列を指定します．

固定長文字列変数に対し，指定された長さより長い文字列を代入すると，残った部分は捨てられてしまいます．

■文字列の格納イメージ

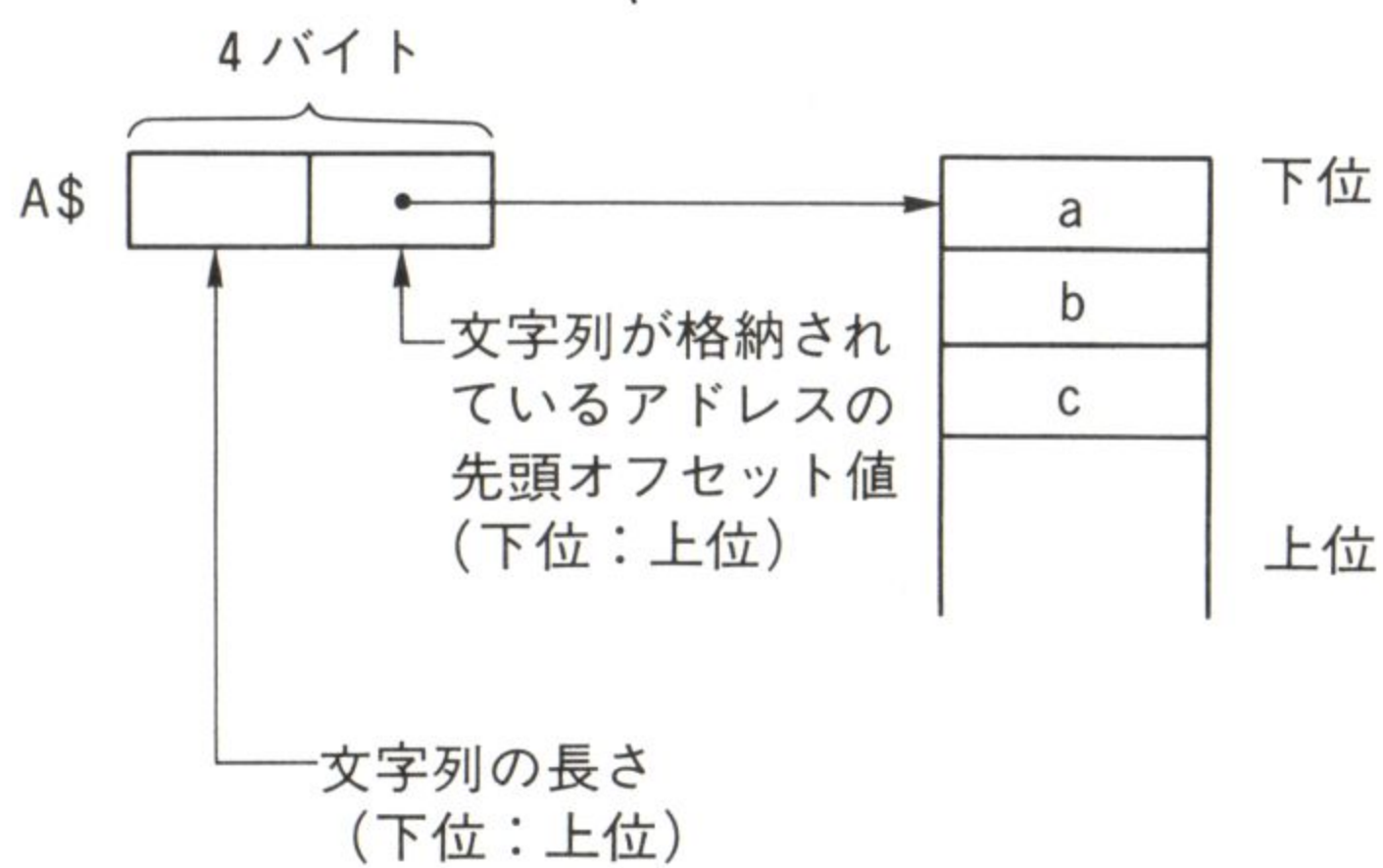
文字列データのメモリ上への格納イメージは，可変長文字列と固定長文字列とで異なります．

たとえば，

```
A$= "abc"
```

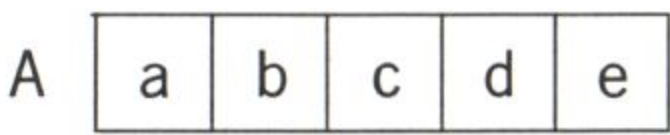
のような可変長文字列は次のようにメモリ上に格納されています．

4バイト
A$
文字列が格納されているアドレスの先頭オフセット値（下位：上位）
文字列の長さ（下位：上位）

a	下位
b	
c	
	上位

つまり，可変長文字列変数は4バイトのサイズで，先頭2バイトで文字列の長さ，後部2バイトで，文字列が格納されているアドレスの先頭オフセット値を示しています．つまり，変数 A$ がある場所と，実際に文字列が格納されている場所とが異なるわけです．

これに対し，固定長文字列では次のように変数の位置に文字列が格納されます．

```
DIM  A  AS  STRING * 5
A = "abcde"
```

A	a	b	c	d	e

5 配列

配列は同種のデータを配列名と添字(要素番号)で管理するデータ構造の1つです．

配列の添字が1つのものを1次元配列，2つのものを2次元配列，以下，3次元，4次元……と呼びます．

■配列の宣言

配列の宣言はDIMを用いて次のように行います．

```
DIM 配列名( 添字の最大値，添字の最大値，…)，…[AS  型名]
```

配列の要素は通常0番の要素から始まりますが，OPTION BASE 1と宣言することで1番の要素から始めることもできます．OPTION BASE 文はすべての配列に対して作用します．

「AS　型名」を省略すると，単精度実数型の配列とみなされます．

例

```
            ┌──────── 添字の上限を宣言する
            ↓
DIM A(10)
```

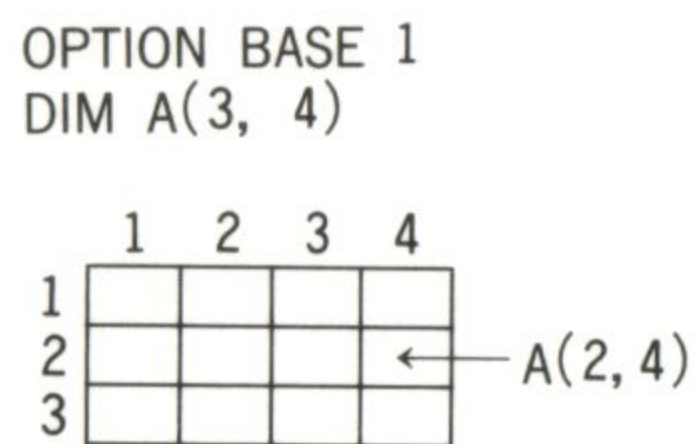
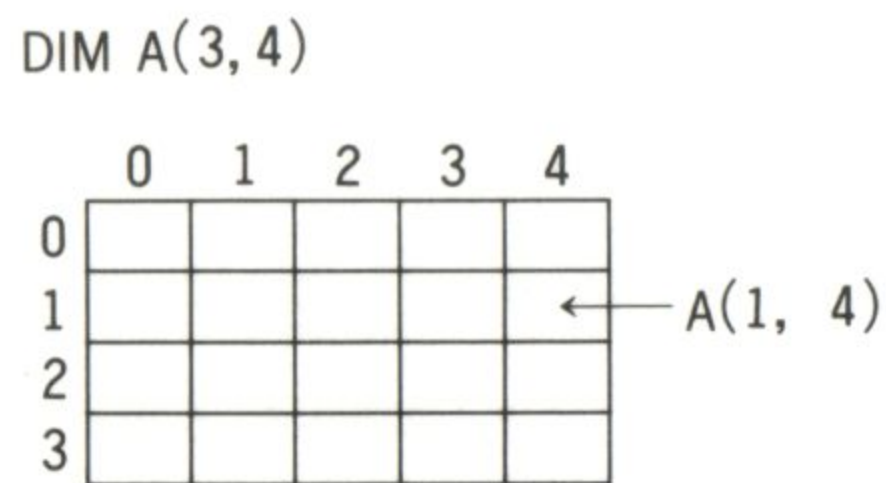
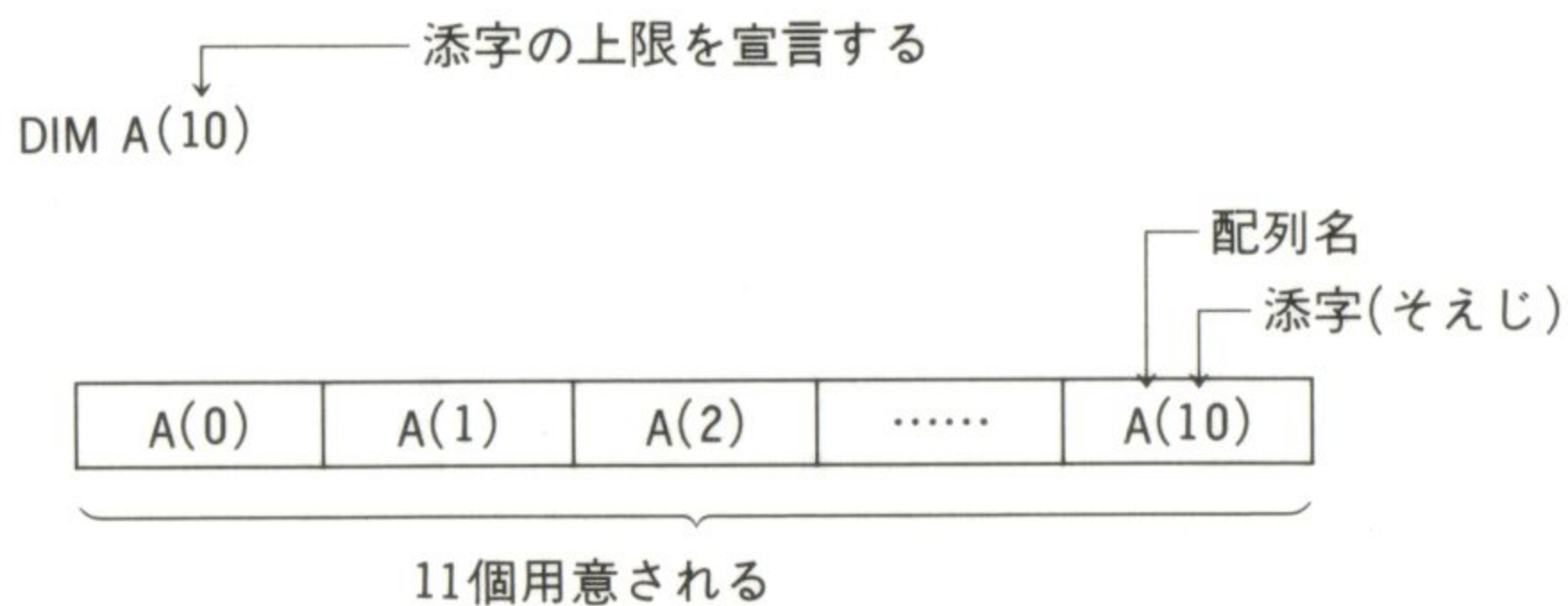

A(0)	A(1)	A(2)	……	A(10)

11個用意される

```
DIM A(3,4)
```

	0	1	2	3	4
0					
1					← A(1, 4)
2					
3					

```
OPTION BASE 1
DIM A(3, 4)
```

	1	2	3	4
1				
2				← A(2,4)
3				

```
DIM Sum(100),A(3,4)   AS   LONG
```

…配列 Sum( ),A(,)は倍長整数型配列として宣言されます.

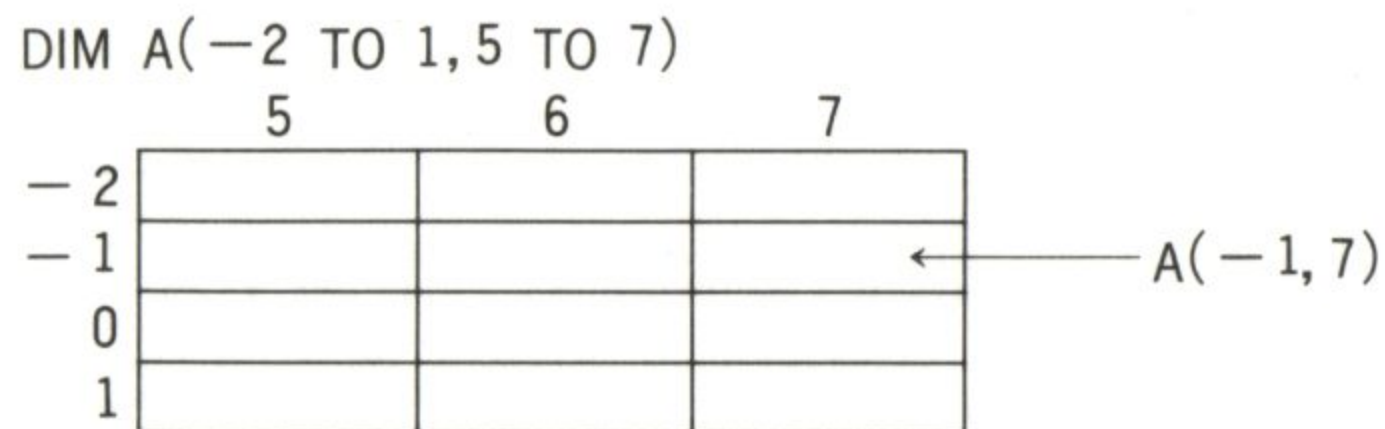
■部分範囲指定

Quick BASIC では配列の添字に部分範囲を指定することができます.

DIM 配列名(最小値 TO 最大値, 最小値 TO 最大値, …)

例

```
DIM A(-2 TO 1,5 TO 7)
```

	5	6	7
−2			
−1			← A(−1,7)
0			
1			

■配列における留意事項

配列の添字の範囲およびサイズについては以下のような制約があります.

- 配列の添字には, −32768〜32767の値が使えます.
- 1次元あたりの要素の数は最大で32768個までです.
- 1つの配列の次元の上限は60次元までです.
- 静的配列の最大サイズは64KB(キロバイト)までです.
  注)静的配列と動的配列の違いについては本章1-8を参照してください.

配列が必要とするメモリ量は，〈配列の要素数〉×〈その型のサイズ〉です．たとえば，

```
DIM  A(255)  AS  LONG
```

という配列のサイズは，256×４＝1024バイト＝1KB(キロバイト)となります．
配列名は，単純変数の名前とは別物として扱われます．たとえば，

```
DIM  A(100)
DIM  A   AS   INTEGER
```

のA(0)～A(100)とAは別物です．しかし，このように配列名と単純変数名に同じものを使うとプログラムがわかりにくくなるので，このような使用はさけるべきです．

6 レコード型

配列は同じ型のデータをひとかたまりにしたものですが，レコード型は異なる型のデータをひとかたまりにしたものです．
下表のデータは，Galileo(ガリレオ) が1610年に発見した木星の４大衛星の衛星名，光度，周期に関するものです．

木星の４つの衛星

衛星名	光度〔等〕	周期〔日〕
Io	5	1.7691
Europa	6	3.5512
Ganymede	5	7.1545
Callisto	6	16.6890

1件のレコードとして扱う

こうしたデータは，衛星名，光度，周期をひとかたまりのデータとして扱えたら便利です．このようにいくつかの項目をひとかたまりにしたものを「レコード」と呼び，各項目を「メンバ(フィールド)」と呼びます．レコード型はこのようなデータを定義するのに便利です．

■レコード型の定義

レコード型はTYPE~END TYPE文を用いて次のように行います．

```
TYPE  タグ名
      メンバ名  AS  型名
      メンバ名  AS  型名
        ⋮
END  TYPE
```

これにより〈タグ名〉で示される新しい型が定義されたことになります．このようにユーザが定義する型を「ユーザ定義型」といいます．

各メンバを指定する型名は次のものです．

INTEGER	(単精度)整数型
LONG	倍長整数型
SINGLE	単精度実数型
DOUBLE	倍精度実数型
STRING ＊長さ	文字列型(固定長)

メンバに指定できる文字列型は，固定長だけで，可変長文字列は認められません．

例

先の表(木星の4つの衛星)のようなデータを扱うレコード型を定義するには次のようにします．

```
            ┌─タグ
TYPE    Eisei               ┌─メンバの型
   Star    AS   STRING   ＊ 10
   Kodo    AS   INTEGER
   Shuki   AS   SINGLE
END↑TYPE
   └──────────メンバ
```

これにより，Eisei 型という新しい型が定義されたことになります．

Eisei をタグ，Star，Kodo，Shuki をメンバと呼びます．タグは変数を宣言するための金型のような働きをします．

■レコード型変数の宣言

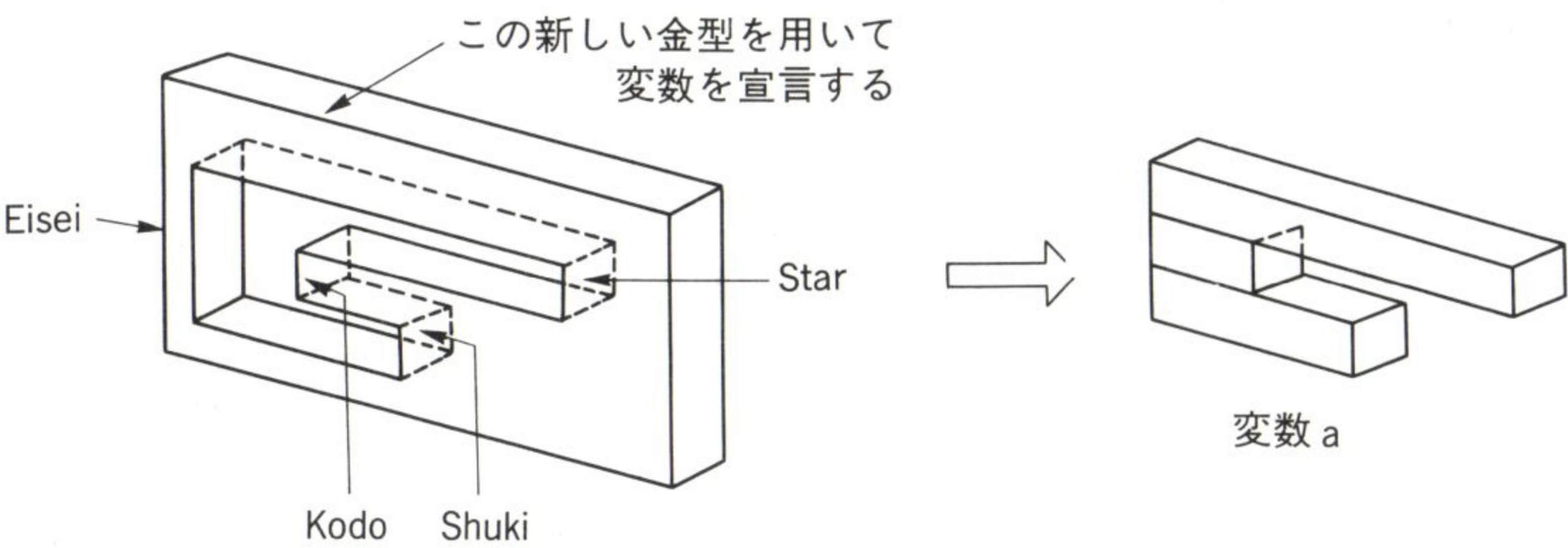
TYPE～END TYPE 文で定義したレコード型のタグ名を用いて，変数を宣言するには，

DIM　変数名，変数名…　AS　タグ名

とします．

例

```
DIM  a  AS  Eisei
```
… Eisei 型の変数 a を宣言

この新しい金型を用いて
変数を宣言する
Eisei
Star
Kodo
Shuki
変数 a

■メンバの参照

レコード型変数の各メンバを参照するにはピリオド(.)を用いて，

レコード型変数名．メンバ名

とします．たとえば，衛星 Io(イオ)のデータをレコード型変数 a に代入するには，次のようにします．

```
a.Star= "Io"
a.Kodo= 4
a.Shuki=1.7691
```

レコード型データは，ひとかたまりにして代入できますので，DIM　b　AS　Eisei と宣言されているレコード型変数 b に対し，

```
b = a
```

とすることで，a の内容は一括で b に代入されます．

7 通用範囲(スコープ)

変数(あるいは記号定数)の内容を参照できる範囲を，スコープ(通用範囲：有効範囲)と呼びます．

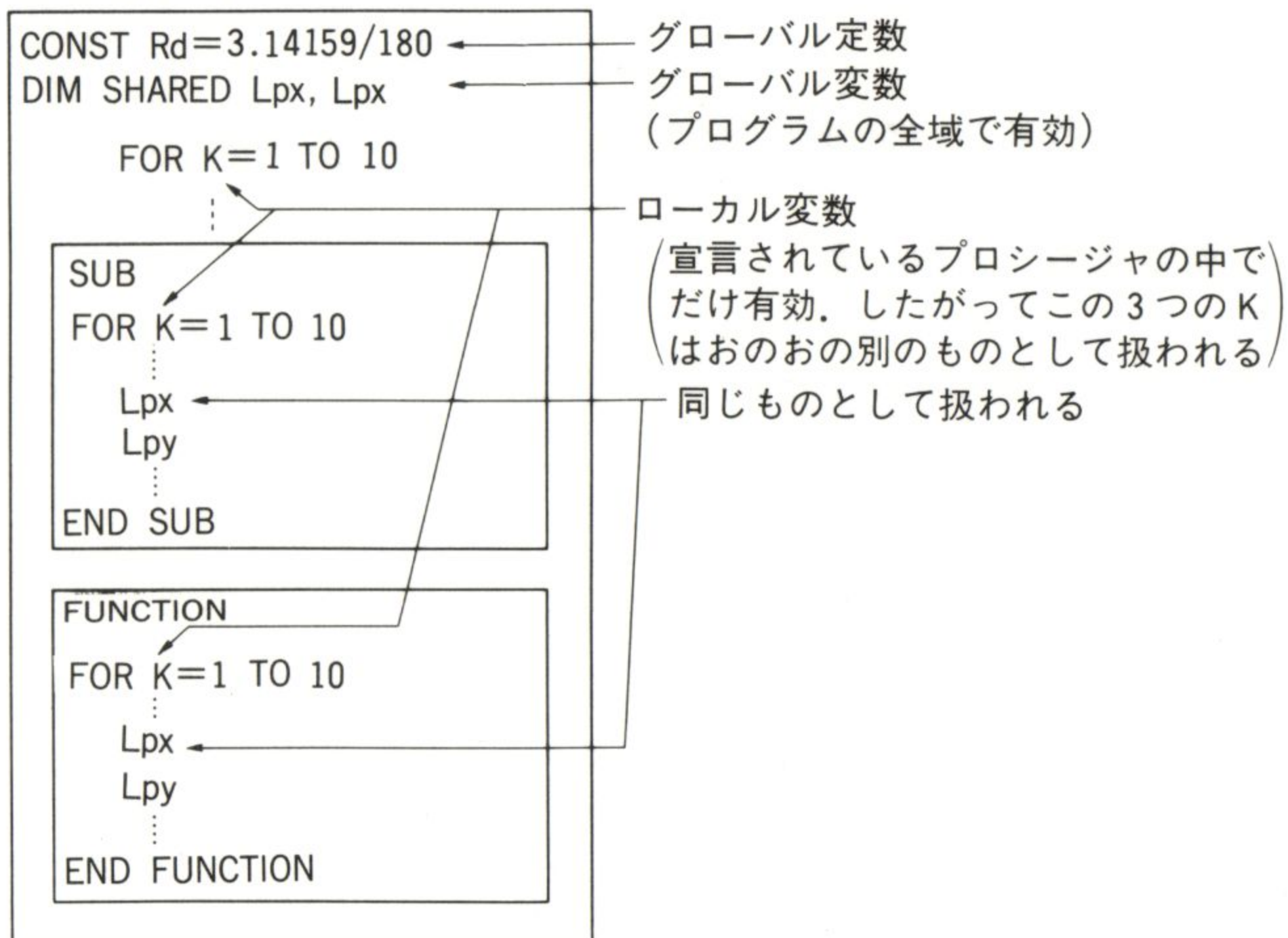
スコープは，グローバル(大域的)とローカル(局所的)の2種類に分けられます．グローバルなスコープはプログラム中のどの部分からも参照できますが，ローカルなスコープはそれが宣言(使用)されているプロシージャの中でだけ参照することができます．

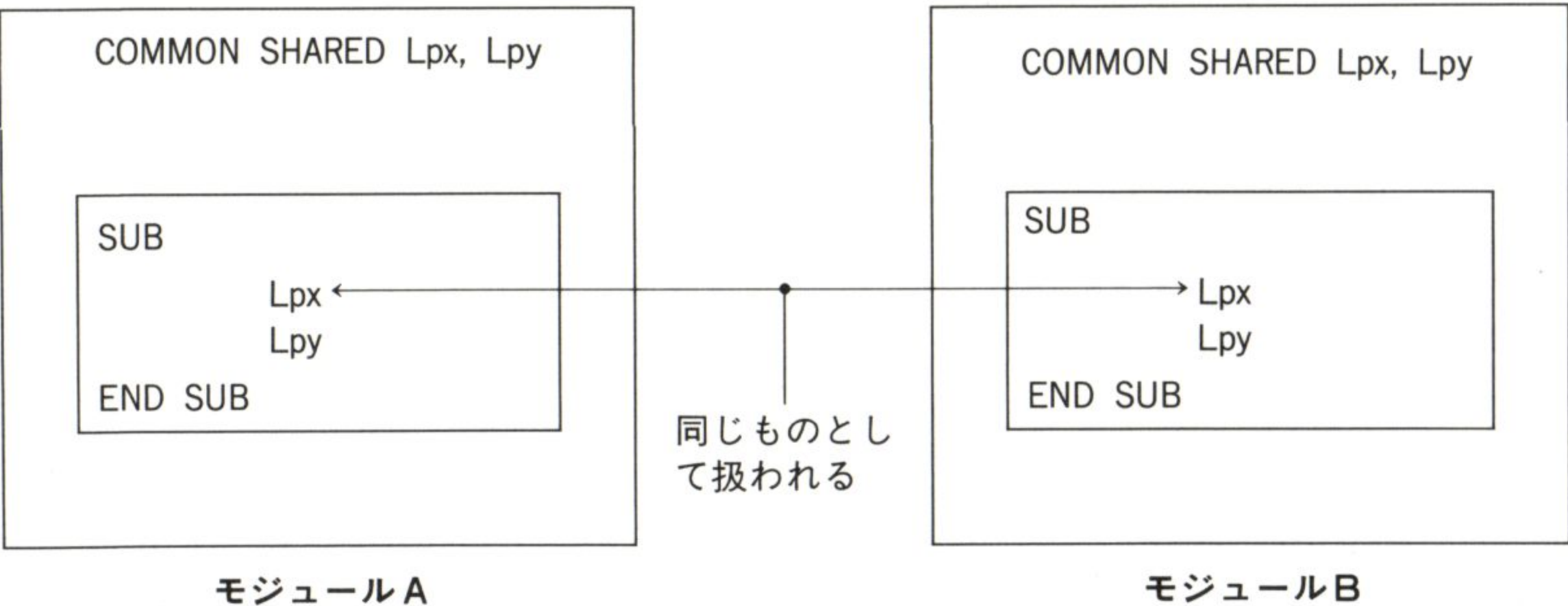
■ローカル変数とグローバル変数

従来型BASICの変数はすべてグローバル変数でしたが，Quick BASICではグローバル変数とローカル変数の2種類が指定できます．

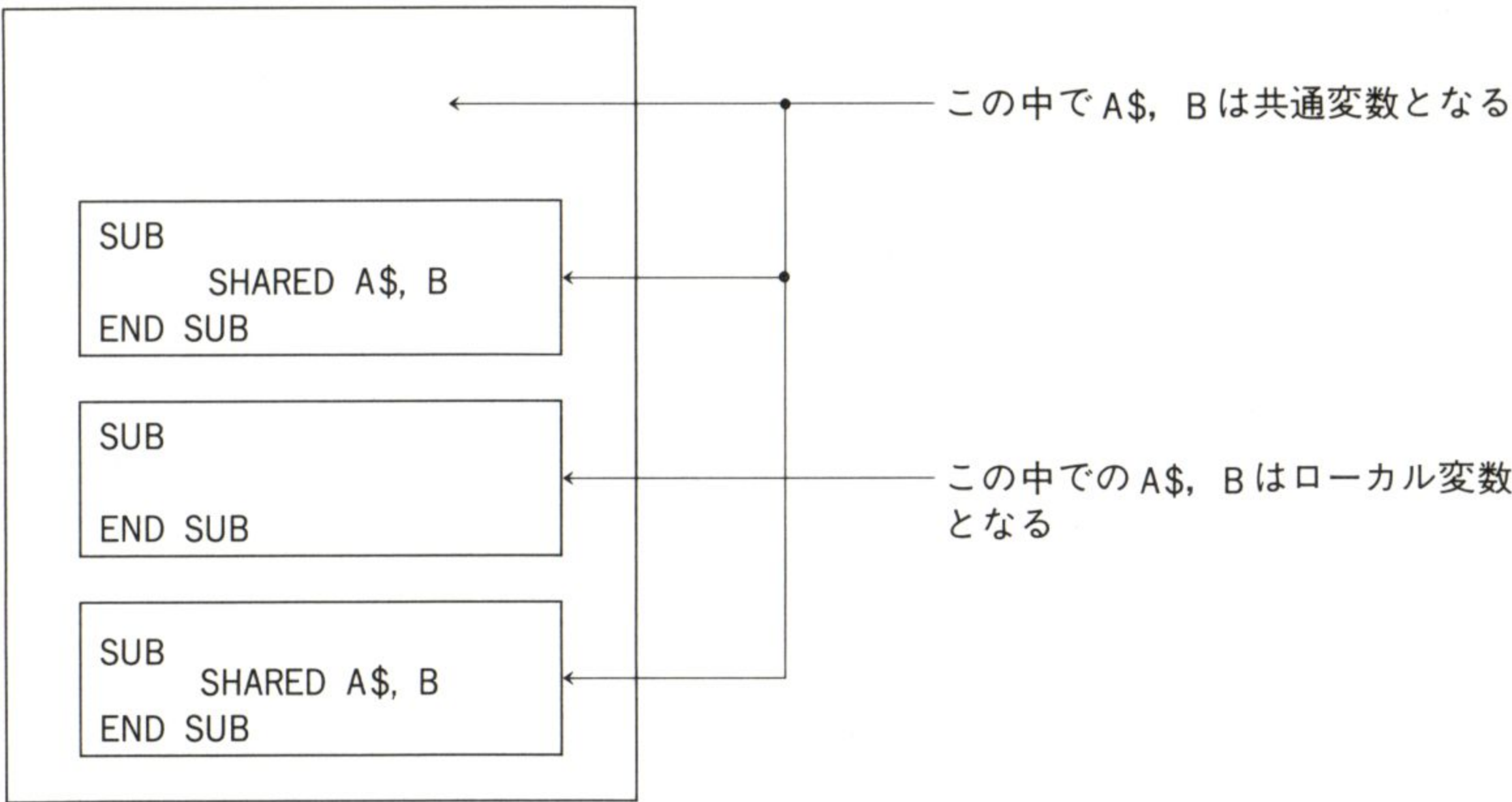
- 一般的な変数はローカル変数です．ローカル変数は，それが宣言(使用)されているプロシージャの中でだけ有効です．
- メインルーチンにおいて，DIM SHAREDで宣言された変数はグローバル変数です．グローバル変数はプログラムの全域で有効です．
- メインルーチンで宣言された記号定数はグローバル定数です．SUBあるいはFUNCTIONプロシージャの中で宣言された記号定数はローカル定数です．

●プログラム

```
CONST Rd=3.14159/180
DIM SHARED Lpx, Lpx

    FOR K=1 TO 10
    ⋮
  SUB
  FOR K=1 TO 10
    ⋮
    Lpx
    Lpy
    ⋮
  END SUB

  FUNCTION
  FOR K=1 TO 10
    ⋮
    Lpx
    Lpy
    ⋮
  END FUNCTION
```

■モジュール間の変数の共有

DIM SHAREDで宣言された変数は，それが宣言されているモジュール内のすべてのプロシージャでグローバルです．つまり単一モジュール内での扱いとなります．これに対し，COMMON SHAREDで宣言された変数は，異なるモジュールのすべてのプロシージャでグローバルです．

COMMONで変数を宣言する場合，変数の宣言順序は，すべてのモジュールの宣言において同一に行わなければなりません．したがって，COMMONの宣言文をインクルードファイルにしておき，各モジュールでは，このファイルをインクルードするようにすると安全です．

■プロシージャ間の変数の共有

SHARED文(DIMやCOMMONのときのSHARED指定とは異なる)を使えば，単一モジュール内の複数のプロシージャにおいて変数を共有することができます．

■ DEF FN 関数内のスコープ

DEF FN 関数はプロシージャではないので，その中で使用されている変数(引数は除く)は，メインルーチンの変数と同一のものです．DEF FN 関数内の変数をローカルにするには，STATIC 文を用います．

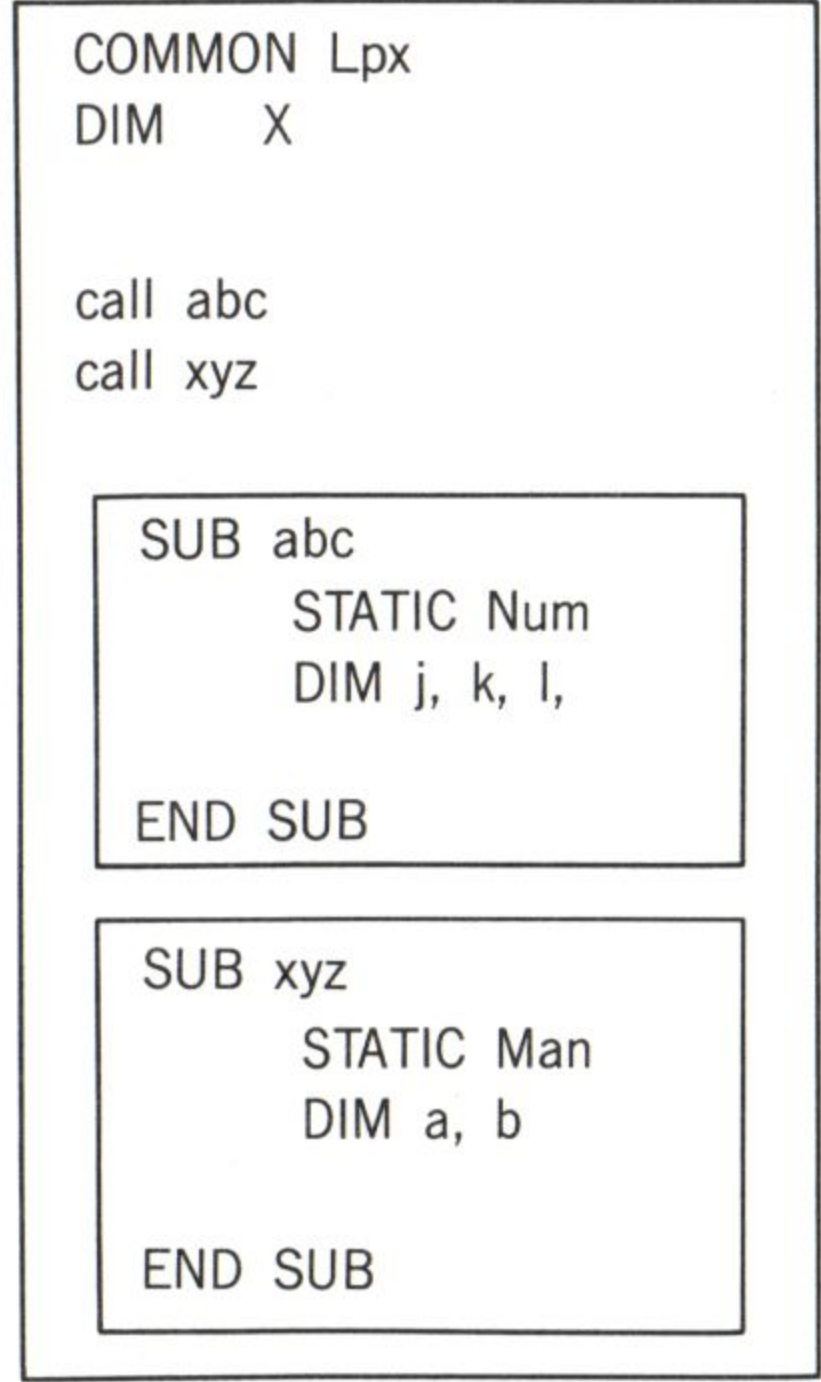
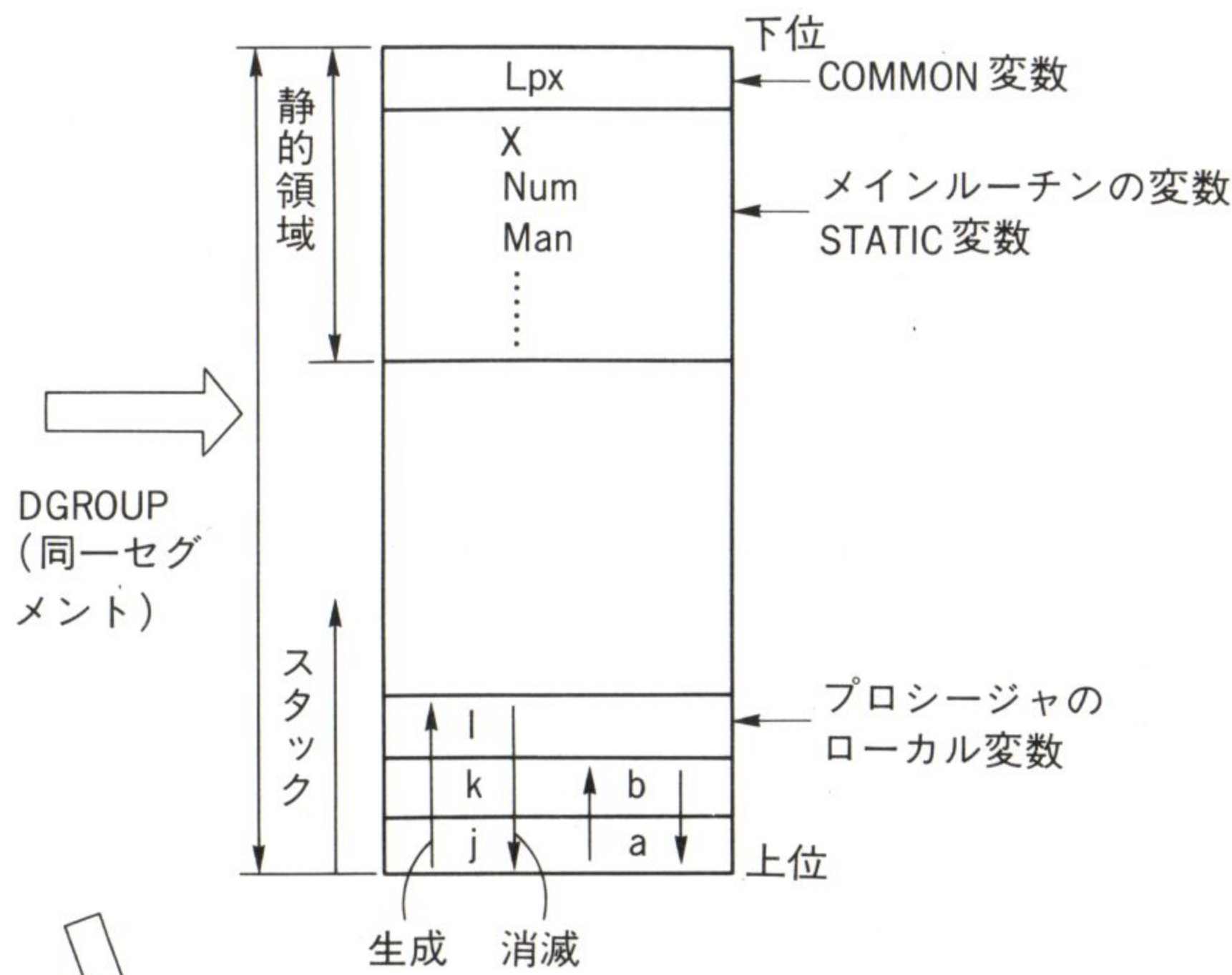
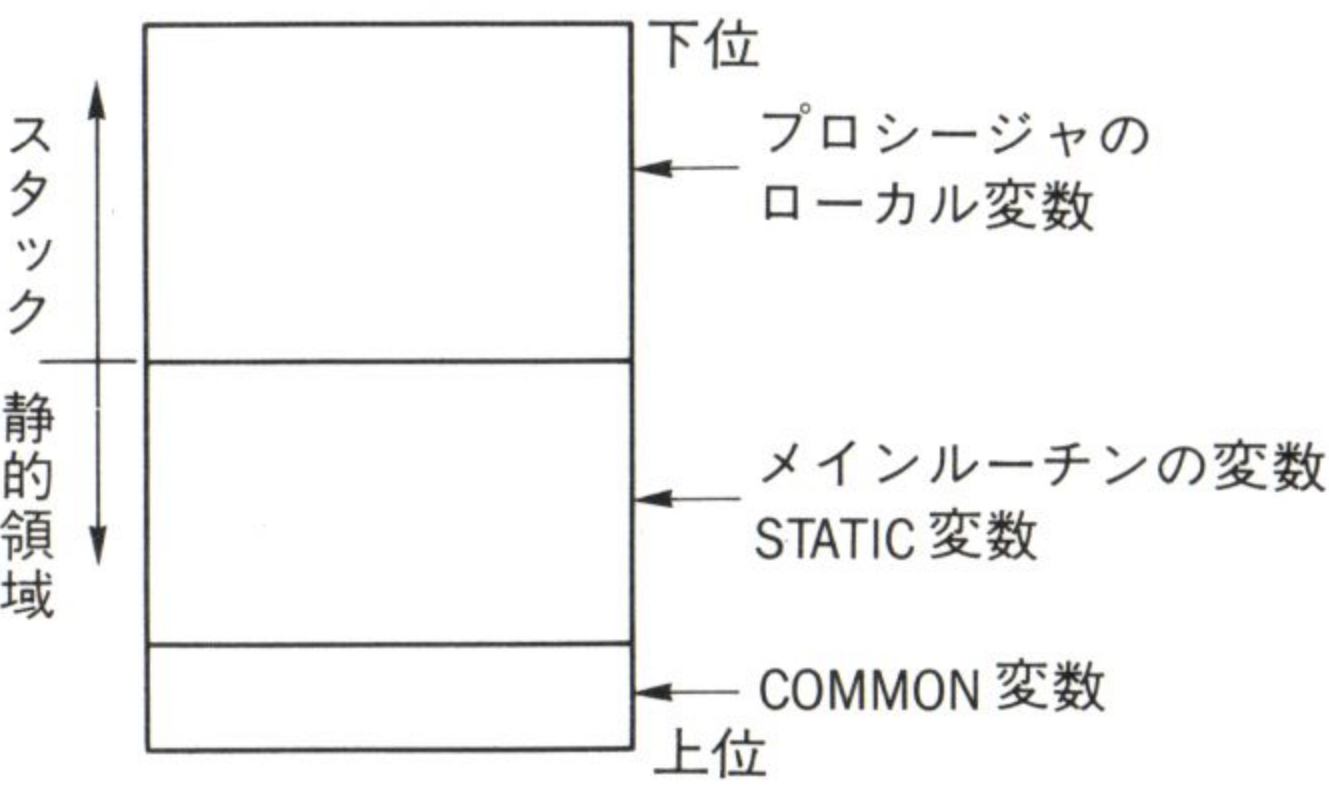
```
DEF FNAbc(x, y)
STATIC i  ←――――――― ローカル変数の指定
    FOR i=0  TO 10
     ⋮

END DEF
```

8 記憶クラス

変数をメモリのどのような領域に割り当てるかで記憶クラスが決まります．メモリ領域は静的領域とスタック領域に大別されます．メインルーチンの変数や，プロシージャの中でSTATIC宣言した変数は，メモリの静的領域に格納され，プログラムの開始から終了まで同じアドレスに割り当てられています．これに対し，プロシージャのローカル変数は，メモリのスタック領域に割り当てられ，プロシージャの開始で生成され，プロシージャの終了で消滅します．

なお，変数のメモリ格納イメージは，.EXEファイルにした場合と，QB環境下の場合とでは下図のように多少異なりますので注意してください．

●プログラム

```
COMMON Lpx
DIM    X

call abc
call xyz

SUB abc
      STATIC Num
      DIM j, k, l,

END SUB

SUB xyz
      STATIC Man
      DIM a, b

END SUB
```

●メモリ格納イメージ(.EXEファイル)

●メモリ格納イメージ(QB環境下)

■静的変数と自動変数

メモリの静的領域に割り当てられている変数を静的変数と呼び，メモリのスタックに割り当てられている変数を自動変数と呼びます．

静的変数は，プログラムの開始から終了まで同じアドレスに確保され，消滅することがないので，プロシージャへの再入時に，前の値を保存していることになります．

自動変数は，それが宣言(使用)されているプロシージャが呼び出されたときにゼロまたはヌル(空文字列)に初期設定されます．

次の例は静的変数を用いて，あるイベントが起こった回数をカウントするものです．

```
:
IF　ある条件　　THEN　call　Count
:

SUB　Count
  STATIC　Num
  Num　=Num　+ 1
  :
END　SUB
```

Num は STATIC 変数なので，Count が呼ばれるたびに前の Num の値を＋1していきます．もし，Num を STATIC 指定しなければ，Count が呼ばれるたびに Num は 0 に初期設定されてしまいます．

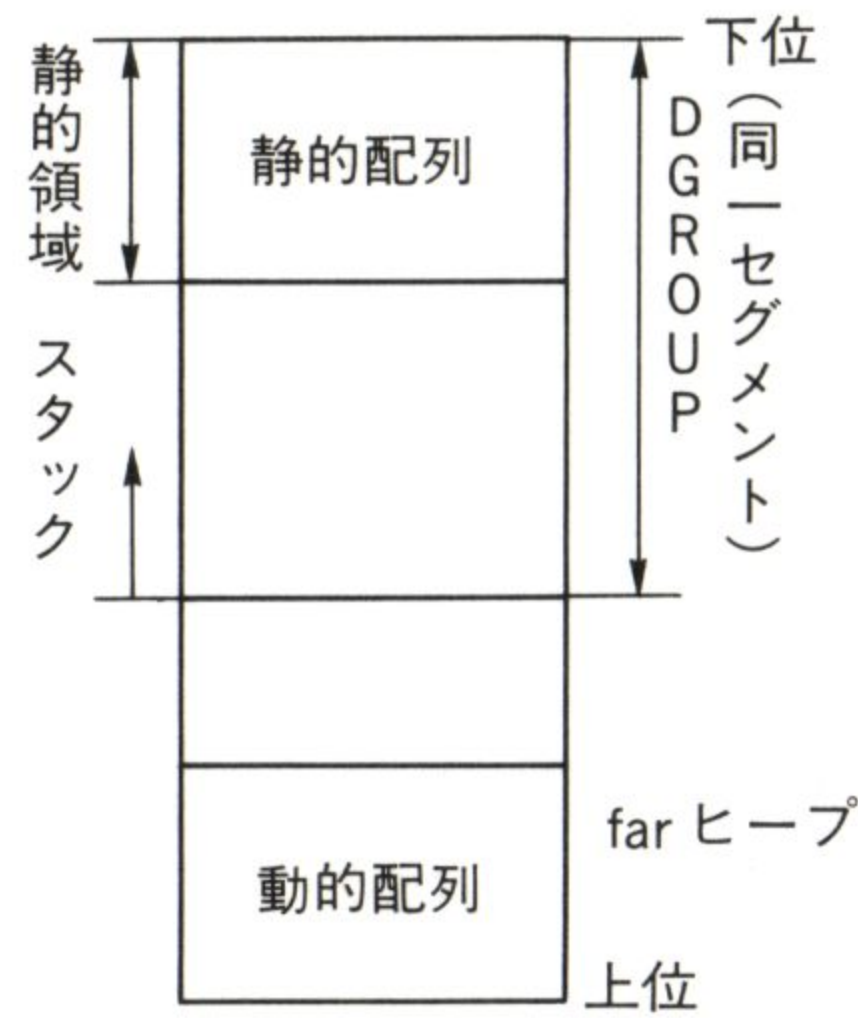
なお，次のようにプロシージャ全体に STATIC 属性を与えることで，すべてのプロシージャ内の変数を STATIC 変数にすることもできます．

```
SUB　プロシージャ名(引数，…)　STATIC
:
END　SUB
```

■動的配列

プログラムのコンパイル時に記憶領域を与えられる配列を静的配列と呼び，プログラム実行時に領域を与えられる配列を動的配列と呼びます．

宣言時に定数の添字を指定したものが静的配列になり，変数の添字を指定したもの，または最初にCOMMON文で宣言されている配列は動的配列になります．

```
DIM  a(100)      ←  静的配列

N=50
DIM  b(N)        ←  動的配列

COMMON  c( )
DIM  c(50)       ←  動的配列
```


原則的には，静的配列はDGROUPセグメントの静的領域にとられ，動的配列はfarヒープ領域にとられますが，配列のメモリ割り当ては.EXE形式にした場合と，QB統合開発環境下の場合とでは以下のように異なります．

●.EXE形式の場合

・動的配列はfarヒープ領域
・静的配列はDGROUP
・可変長文字列の配列は静的配列，動的配列ともDGROUP

● QB環境下

・COMMON静的配列，すべての可変長文字配列はDGROUP
・これ以外の配列はfarヒープ領域

なお，配列を動的配列にするか静的配列にするかは，$DYNAMIC，$STATICメタコマンドによっても指定することができます．

9 混合演算と型変換

数値データは，異なる型同士でも混合して演算することができます．こうした混合演算では精度の高い型に変換されて演算が行われます．たとえば，

```
A#=10#/3
```

は，10#/3#として演算が行われます．

なお，型変換は式の中の各演算単位で行われていくので，

```
x%=1 :  y!=3 :  z#=3
PRINT   x% / y! * z#
```

のような式では，まず x% / y! が単精度実数型として演算され，その結果が倍精度に変換され z# と乗算されますから，結果は1.0 とはなりません．

また，数値データが異なる型の変数に代入されるときは，次のような規則で変換されます．

・整数型変数←実数データ

実数データが整数に変換される場合は，小数点以下が四捨五入されます．なお，実数データの整数部が－32768～32767の範囲を越えていればエラーが起こります．

```
A%=34.4        A%は34
B%=34.5        B%は35
```

・実数型変数←整数データ

整数データが実数型に変換される場合は，.0 が付加されます．

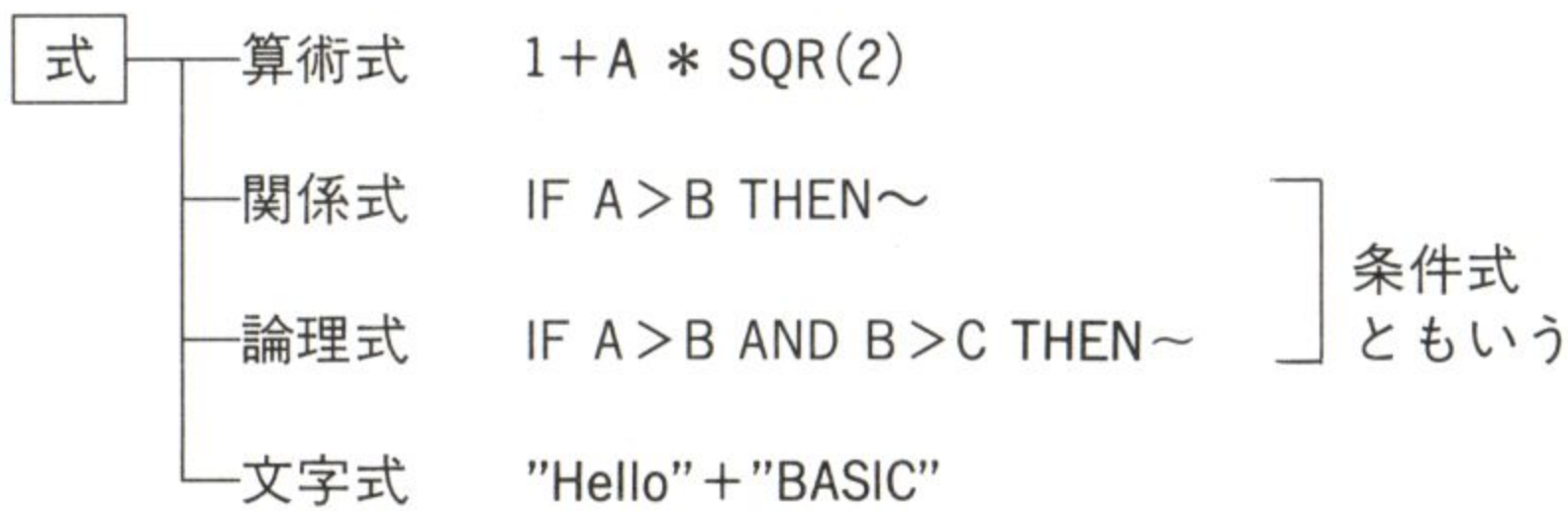
・単精度実数型変数←倍精度実数データ

倍精度実数データが単精度実数型に変換される場合は，有効桁数が7桁にまるめられます．

4－3 式と演算子

1 式

定数，変数，関数，およびこれらを演算子で結び付けたものを式といい，次のような種類があります．

式	種類	例	
式	算術式	1＋A ＊ SQR(2)	
	関係式	IF A＞B THEN～	条件式ともいう
	論理式	IF A＞B AND B＞C THEN～	条件式ともいう
	文字式	”Hello”＋”BASIC”	

2 演算子の種類と優先順位

Quick BASIC の演算子の種類と優先順位を以下に示します．

	演　算　子	優先順位
	かっこで囲まれたもの	1
	関数	2
算術	^	3
	－(負記号)	4
	＊，／	5
	¥	6
	MOD	7
	＋，－	8
関係	＝　＜＞　＜　＜＝　＞　＞＝　＞＜　＝＜　＝＞	9
論理	NOT	10
	AND	11
	OR	12
	XOR	13
	EQV	14
	IMP	15

これらの演算子は算術演算子，関係演算子，論理演算子に分類されます．

3 算術演算子

算術演算子として以下のものがあります.

●算術演算子

演算子	意　味	優先順位
^	べき乗	3
−	負符号	4
*	乗　算	5
／	除　算	5
＋	加　算	8
−	減　算	8
¥	整数の除算	6
MOD	整数除算の余り	7

例

10　¥　3

…整数型の除算を行うので結果は 3

10　MOD　3

…10÷ 3 の余りで 1

10　¥　3　*　3

…¥より*のほうが優先順位が高いので 3 * 3 が先に計算され, 10¥ 9 となるので結果は 1

4 関係演算子

条件判断の際に大きい, 小さい, 等しいなどの大小関係を判定する演算子を関係演算子と呼び次の 6 つがあります.

●関係演算子

演　算　子	意　　味	優先順位
＝	等しい	9
＜＞または＞＜	等しくない	9
＜	小さい	9
＜＝または＝＜	小さいか等しい	9
＞	大きい	9
＞＝または＝＞	大きいか等しい	9

比較の結果は，条件を満たせば式の値は真(－1)，満たさなければ偽(0)となります．

一般にIF文の条件式は，条件式の値が0なら偽，非0なら真として判断されます．

実数型のイコール比較は誤差を伴うので避けてください．たとえば，

```
       Sum=0.0
100    Sum=Sum +0.1
         IF  Sum=100.0   THEN  200
       GO TO  100
200
```

のようなプログラムにおいて，Sumの値が正確に100.0になる保証はありませんから，IF文の条件式が一致しない可能性があります．

5 論理演算子

いくつかの条件式を組み合わせるものを論理演算子と呼び，次の6つがあります．

●論理演算子

演算子	意　味	優先順位
NOT	否定	10
AND	かつ	11
OR	または	12
XOR	排他的論理和	13
EQV	同値	14
IMP	包含	15

T(TRUE)を真，F(FALSE)を偽を示すものとして，XとYの論理演算を行ったときに得られる結果を表にしたものを以下に示します．

●真理値表

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
F	F	T	F	F	F	T	T
F	T	T	F	T	T	F	T
T	F	F	F	T	T	F	F
T	T	F	T	T	F	T	T

例

0 <=A AND A<=100

…A が 0 ～100 の範囲に入るかの判定

A<B OR A<C AND A>10

…AND のほうが優先順位が高いので A<C AND A>10 が先に判定される．

■ビット演算

NOT～IMP の論理演算子は内部的にはビット演算を行っています．

また関係式は比較の結果が真なら－1，偽なら 0 となります．たとえば，A>B が真(－1：&HFFFF)，B>C が偽(0：&H0000)の場合，A>B AND B>C は&HFFFF AND &H0000が行われ，ビットごとのANDがとられて結果は&H0000(0)，つまり，偽となります．

条件式の場合は，このような内部的なビット演算は考えず，単に真/偽だけを考えればよいわけですが，NOT～IMP の論理演算子を用いて意図的にビットごとの論理演算を行うこともできます．

ビット演算の真理値表は，先の真理値表の F を 0，T を 1 に読み換えたものになります．

● NOT(否定：ビット反転)

NOT は各ビットを反転します．

たとえば，NOT &hffla は &h00e5 となります．

```
NOT  )1111 1111 0001 1010   …&hffla
      ───────────────────
      0000 0000 1110 0101   …&h00e5
```

● AND(論理積)

AND は特定ビットをマスクするときによく使われます．

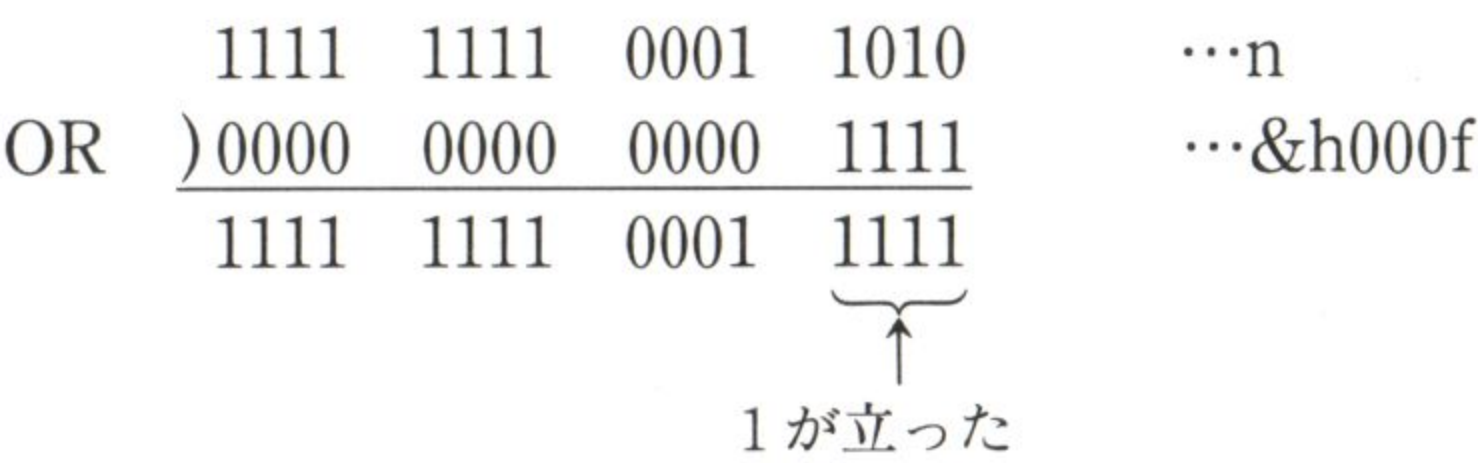
たとえば，n AND &h7fff は n の最上位(第15)ビットを強制的に‘0’にします．

```
      1111 1111 0001 1010   …n
AND  )0111 1111 1111 1111   …&h7fff
      ───────────────────
      0111 1111 0001 1010
      ↑
      0 にマスクされた
```

● OR（論理和）

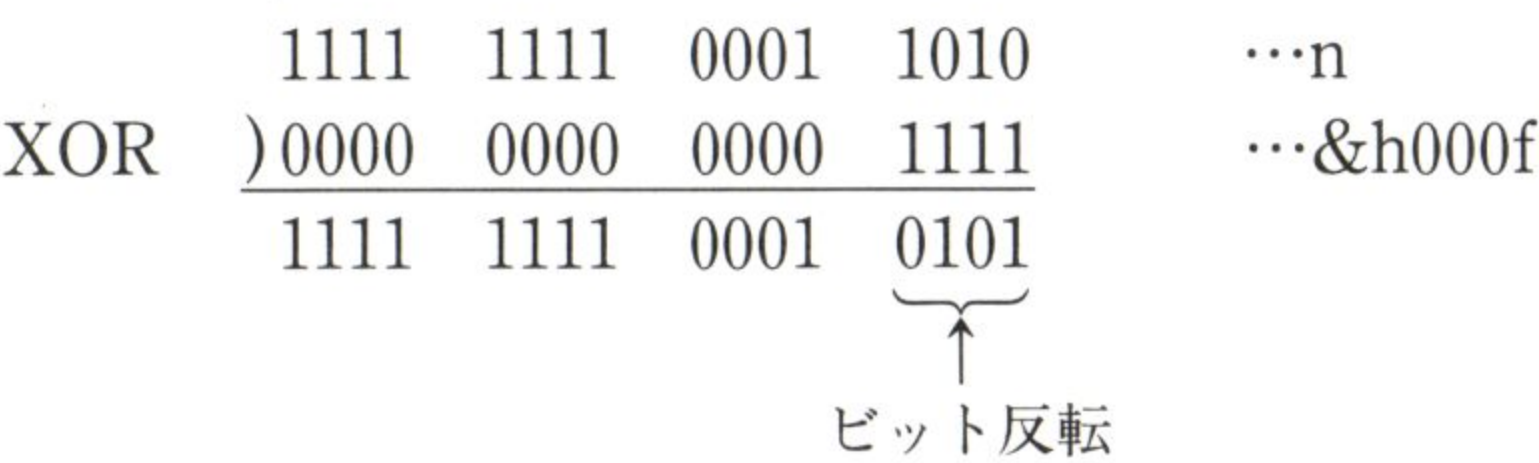
ORは特定ビットを立てるときによく使われます．

たとえば，n OR &h000f はnの下位4ビットを強制的に‘1’にします．

```
      1111  1111  0001  1010      …n
OR   )0000  0000  0000  1111      …&h000f
      1111  1111  0001  1111
                        ↑
                     1が立った
```

● XOR（排他的論理和）

XORは特定ビットを反転するときによく使われます．

たとえば，n XOR &h000f はnの下位4ビットをビット反転します．

```
      1111  1111  0001  1010      …n
XOR  )0000  0000  0000  1111      …&h000f
      1111  1111  0001  0101
                        ↑
                     ビット反転
```

6 文字列演算

文字列演算として，連結と比較が行えます．

文字列は演算子＋によって連結することができます．

```
A$＝ "Hello"＋" Basic"    → A$ は "Hello Basic"
A$＝ "Hello" : B$＝ "Basic"
C$＝A$＋"␣"＋B$          → C$ は "Hello Basic"
```

文字列の大小はアルファベット順(辞書順)に行われます．正確にいうなら，文字の大小はその文字のアスキー・コードの大小で区別されます．２つの文字列を比較していて，文字列の片方が短くて比較が途中で終わった場合は，短い文字列のほうが小さくなります．

たとえば，次のような大小関係になっています．

```
"A" < "AA" < "AB" < "B" …… "Z" < "a"
```

１つの文字定数および文字変数で扱うことのできる文字列の長さは32767文字以下です．

Quick BASIC では，固定長文字列をサポートします．固定長文字列の比較は注意が必要です．

```
DIM  A  AS  STRING *10
A =“/”
IF  A =“/”  THEN～
```

のような比較ではAの内容が，

／	␣	␣	␣	␣	␣	␣	␣	␣	␣

となっているので，“/”と比較しても一致しません．

4－4　制御構造

1　構造化プログラミングと近代的流れ制御構造

構造化プログラミングは，オランダのダイクストラ(E. W. Dijkstra)が提唱したプログラミングの方法論の１つです．

ダイクストラは，すべてのプログラムは，

- 連接(sequence)
- 判断（if then else)
- 前判定反復（while)

という３つの基本制御構造を組み合わせて書くことができるとしました．これをストラクチャード定理といいます．

しかし，実際にプログラミングする場合は，

- 後判定反復（do～loop)
- 複数条件判断（select case)
- 所定回反復（for)

なども必要になります．以上の６つを近代的流れ制御構造と呼びます．

なお，連接というのは単に実行文が上から下に並んでいるものをいい，特別な制御文があるわけではありません．

従来型 BASIC は，後判定反復，複数条件判断などの制御構造がありませんでしたが，Quick BASIC では近代的流れ制御構造を完備しています．さらにブロックという概念の導入により，IF 文が構造的にスマートに記述できるようになりました．

2 判断

条件によって分岐するもので，単純 IF 文，ブロック IF 文，SELECT CASE 文があります．

■単純 IF 文 （従来型 IF 文）

次のようにすべてを1行に書き，最後に END IF を置かない形式の IF 文を単純 IF 文(従来型 IF 文)と呼びます．

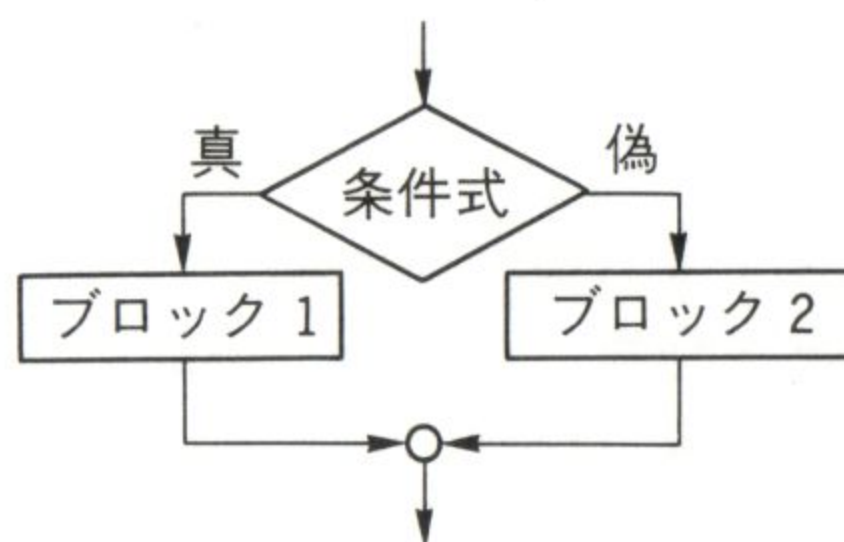
```
IF 条件式 THEN 文1 ELSE 文2
```

文はマルチステートメントで区切れば複数書けますが，1行に収まる範囲です．

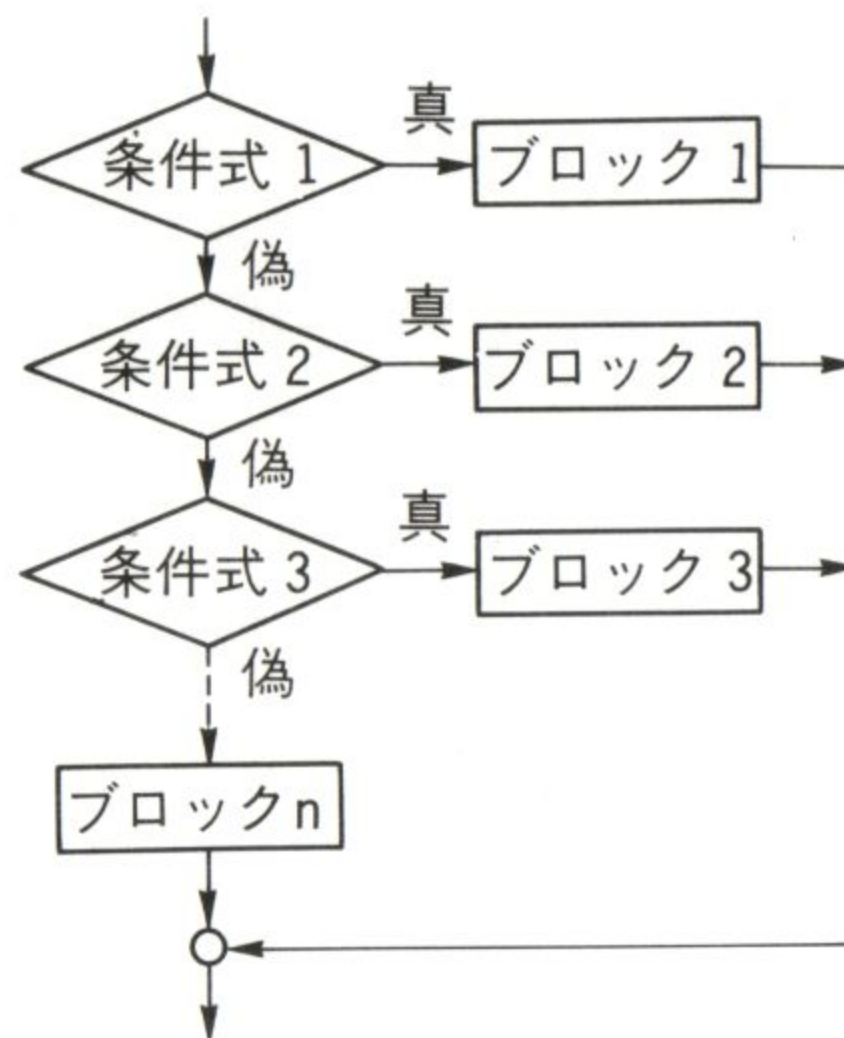
■ブロック IF 文

THEN 節，ELSE 節にブロックを置くことができる IF 文をブロック IF 文といい，Quick BASIC で追加された強力な制御構造です．

いくつかの文を文法的にひとかたまりにしたものをブロック(複文)といいます．

```
IF 条件式 THEN
        ブロック1
ELSE
        ブロック2
END IF
```


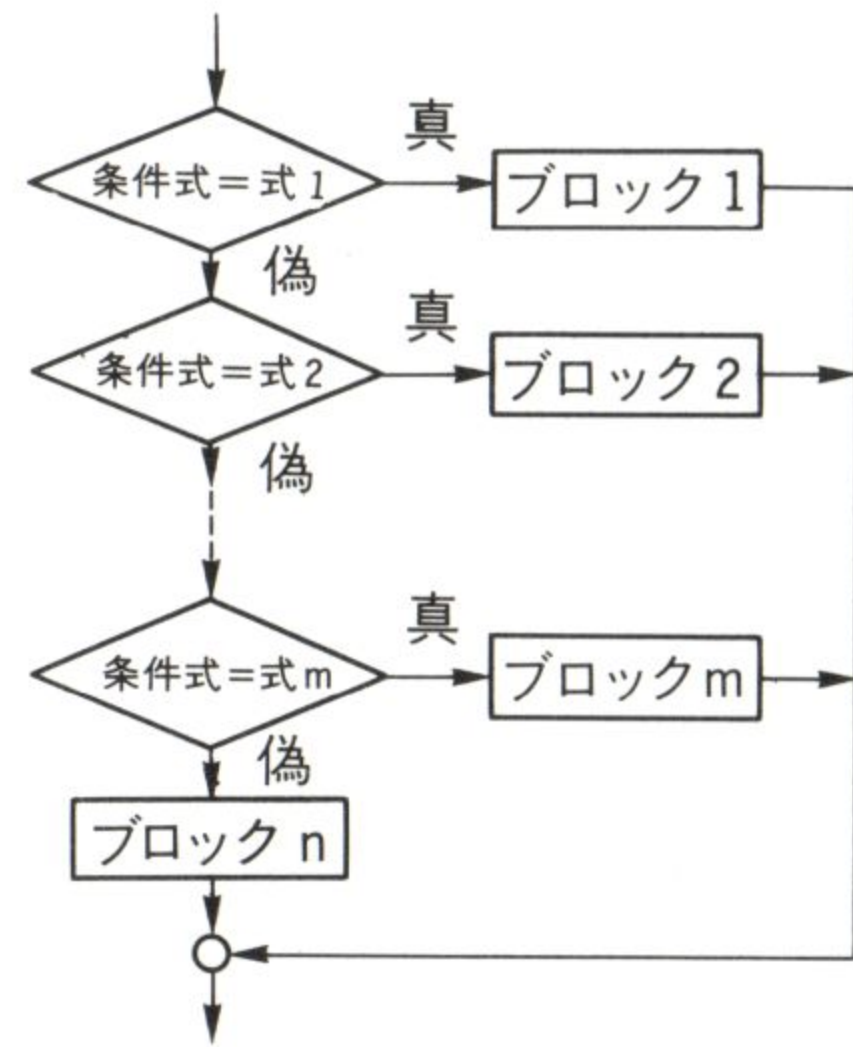
次のように ELSEIF 節を置くことで，複数の条件判断を置くことができます．

```
IF 条件式1 THEN
        ブロック1
ELSEIF 条件式2 THEN
        ブロック2
ELSEIF 条件式3 THEN
        ブロック3
   ⋮
ELSE
        ブロックn
END IF
```


ELSEIF 節や ELSE 節は置くものがなければ省略できますが，END IF は省略できません．

■複数条件判断　(SELECT CASE 文)

ブロックIFのELSE IF節により，複数の条件を判定することができましたが，SELECT CASE文を用いればもっとスマートな複数条件判定をすることができます．

```
SELECT CASE 条件式
      CASE 式1
            ┌─────────┐
            │ブロック1 │
            └─────────┘
      CASE 式2
            ┌─────────┐
            │ブロック2 │
            └─────────┘
        ⋮
      CASE 式m
            ┌─────────┐
            │ブロックm │
            └─────────┘
      CASE ELSE
            ┌─────────┐
            │ブロックn │
            └─────────┘
END SELECT
```


〈条件式〉と一致する式を〈式1〉から順に探し，一致したところのブロックを実行してSELECT CASE文から抜けます．もし一致する式がなければCASE ELSE節のブロックが実行されます．CASE ELSE節はなくてもかまいません．

CASEの後に書く式は，5や“abc”のような値のほかに，次のように値の範囲を指定することができます．

CASE　式1　TO　式2
CASE　IS　関係式

3 くり返し

くり返しを行う制御文として，所定回反復，前判定反復，後判定反復があります．

■所定回反復　(FOR 文)

くり返し回数があらかじめ定まっているくり返しを行います．

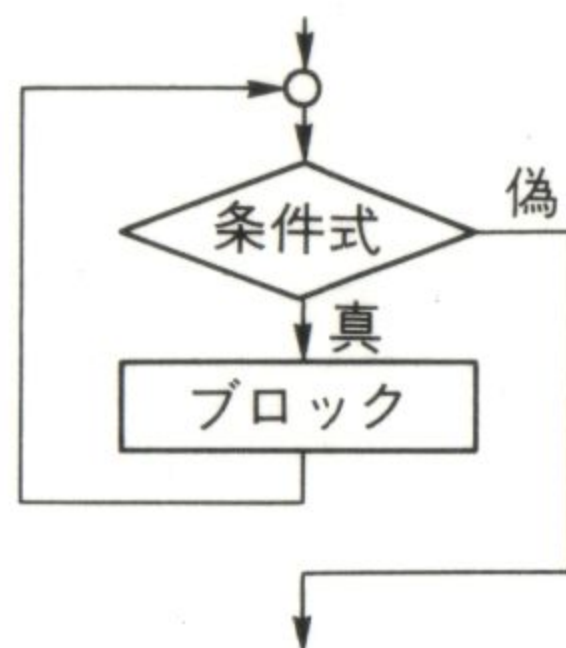
```
FOR ループ変数＝初期値 TO 最終値 STEP きざみ値
      ┌──────────┐
      │文        │
      │⋮         │
      └──────────┘
NEXT ループ変数
```

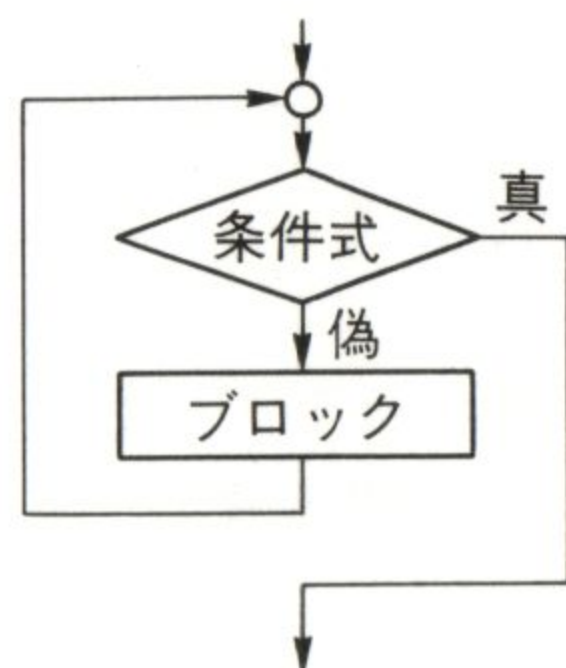
■前判定反復　(WHILE～WEND 文，DO～LOOP 文)

前判定反復は，くり返し回数が定まっていないくり返しを行います．そして，くり返しの終了判定をループの先頭で行います．

前判定反復を行う制御文として DO WHILE～LOOP 文，DO UNTIL～LOOP 文，WHILE～WEND 文の 3 つがあります．

WHILE～WEND と DO WHILE～LOOP は同じ機能です．WHILE～WEND は従来型 BASIC にあったもので，DO～LOOP は Quick BASIC で新たに導入したものです．

```
WHILE　条件式
        ブロック
WEND

DO WHILE　条件式
        ブロック
LOOP
```

〈条件式〉が真の間，〈ブロック〉をくり返します．

```
DO UNTIL　条件式
        ブロック
LOOP
```

〈条件式〉が偽の間(真になるまで)，〈ブロック〉をくり返します．

■後判定反復（DO～LOOP WHILE文，DO～LOOP UNTIL文）

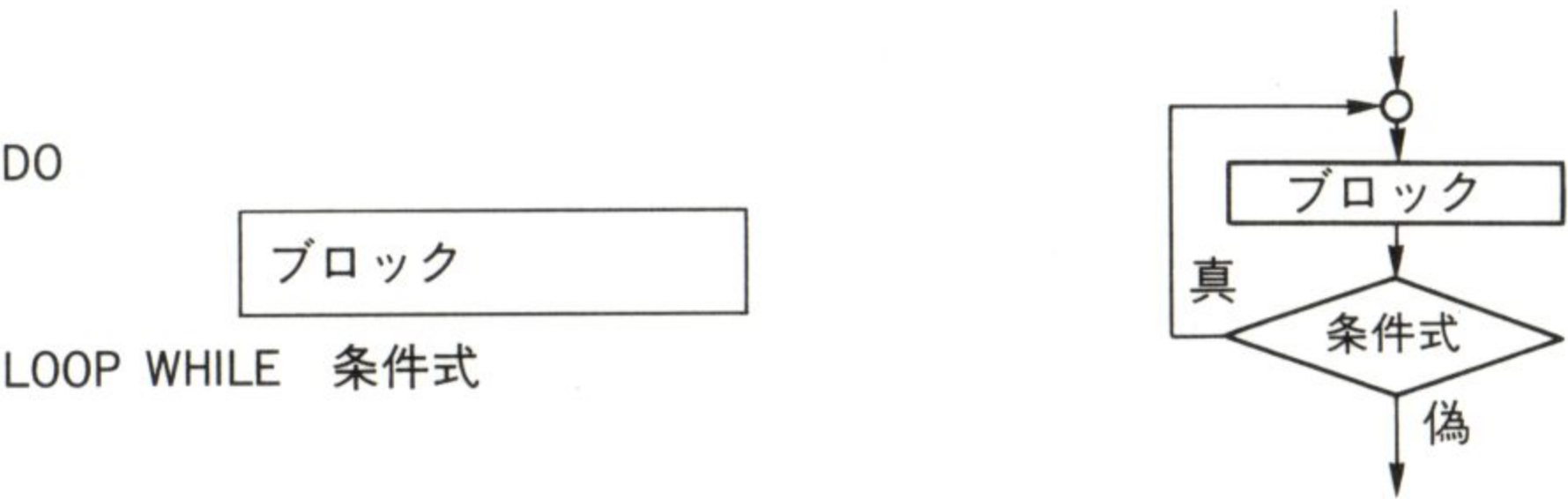
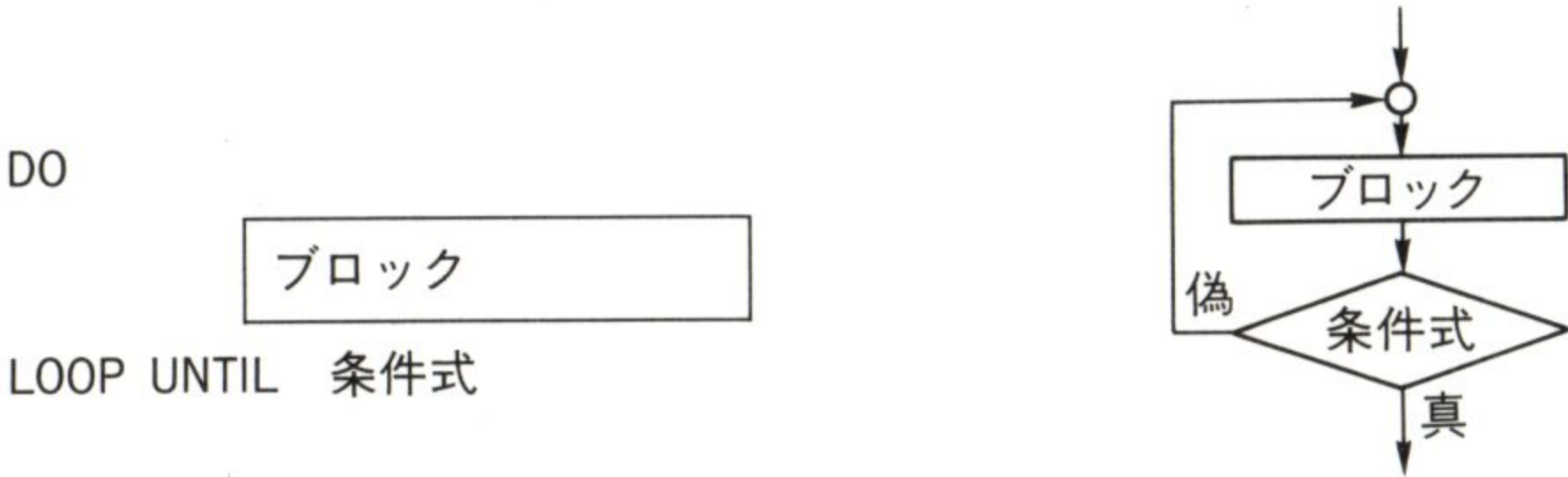
後判定反復は，くり返し回数が定まっていないくり返しを行います．そして，くり返しの終了判定をループの後部で行います．

後判定反復を行う制御文として，DO～LOOP WHILE文とDO～LOOP UNTIL文があります．

```
DO
        ブロック
LOOP WHILE  条件式
```

ブロック
真
条件式
偽

〈条件式〉が真の間，〈ブロック〉をくり返します．

```
DO
        ブロック
LOOP UNTIL  条件式
```

ブロック
偽
条件式
真

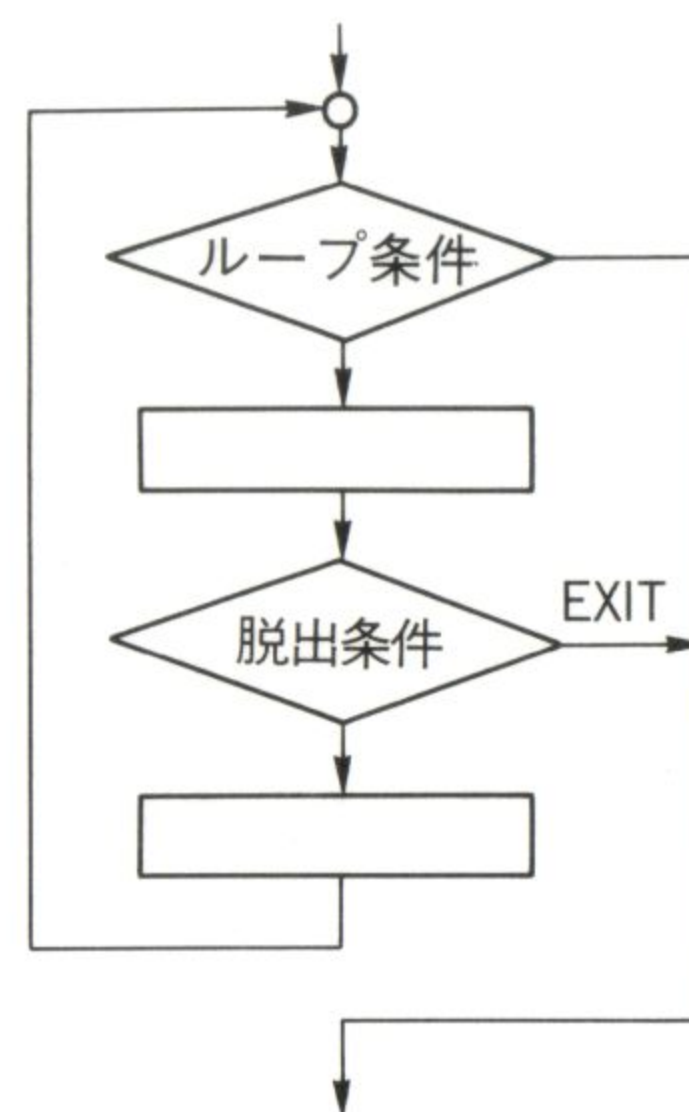
〈条件式〉が偽の間(真になるまで)，〈ブロック〉をくり返します．

4 とび越し

ほかの場所にとび越しを行う制御文として，EXIT 文と GOTO 文があります．EXIT 文はループ中から強制脱出する際によく使いますが，GOTO 文はほとんど使う必要はありません．

■ EXIT 文

EXIT 文は，ループの途中でループから抜ける場合に使います．

```
DO WHILE  条件式
        ⋮
        IF  脱出条件  THEN EXIT DO
        ⋮
LOOP
```


EXIT DO は DO～LOOP から抜けるものですが，この他に EXIT DEF，EXIT FOR，EXIT FUNCTION，EXIT SUB があります．

■ GOTO 文

```
GOTO    OWARI
  ⋮
OWARI:
```

構造化プログラミングという考え方では，GOTO 文はプログラムの流れをわかりにくくするのでできるだけ使わないようにしています．

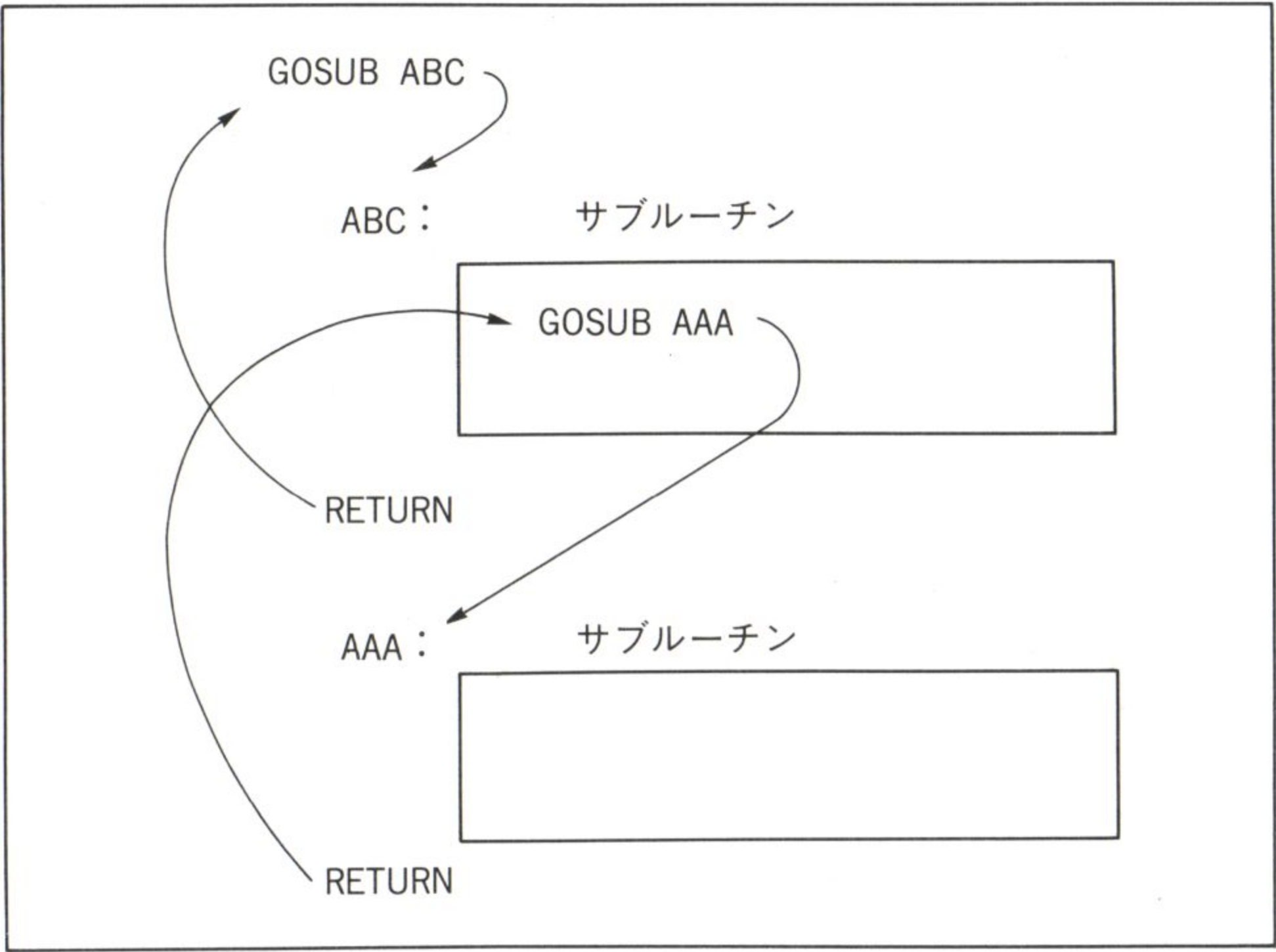
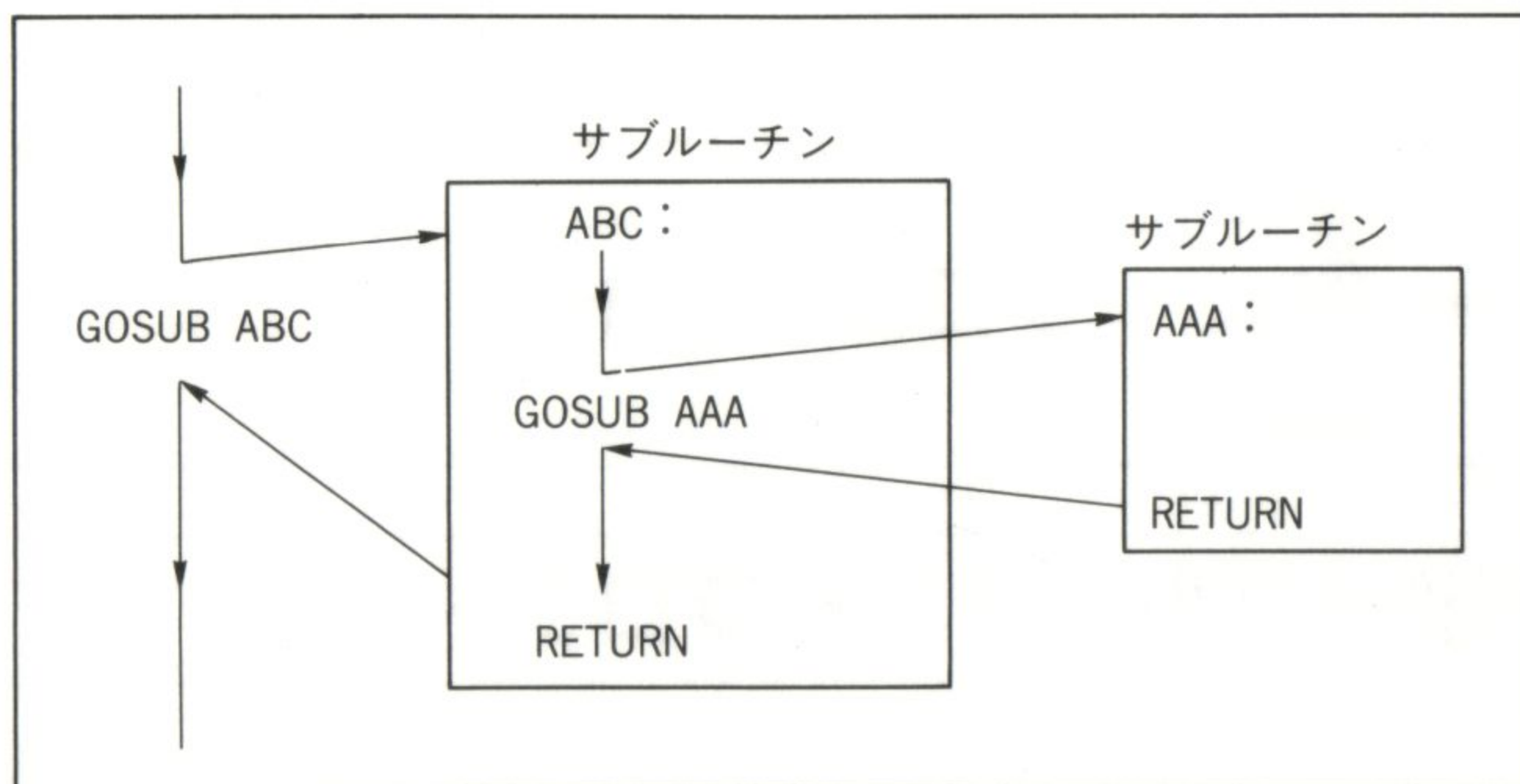
4－5 プロシージャ

1 従来型サブルーチンとプロシージャ

従来型のサブルーチン(GOSUB 文で呼び出すもの)はメインルーチンに従属する一部として構成されます.

サブルーチンは，あるひとかたまりの処理を記述した小プログラムと考えることができます．サブルーチンはGOSUB 文により必要に応じて呼び出すことができます．サブルーチンの入口は行番号またはラベルで管理されます．サブルーチンのRETURN 文が実行されると，GOSUB 文の次の文に戻ってきます．サブルーチンから別のサブルーチンを呼び出すこともできます.

このプログラムが実行されたときの流れを下図に示します.

```
GOSUB ABC

ABC：        サブルーチン
    GOSUB AAA

RETURN

AAA：        サブルーチン

RETURN
```

従来型のBASICで扱っているサブルーチンは，次のような欠点を持っていました．

- サブルーチン間のデータ授受は共通変数を介して行うため，それと異なる変数をサブルーチンに渡す場合は煩雑になる．
- メインルーチンとサブルーチンで使う変数はすべて共通なため，各ルーチン間の独立性が保てない．

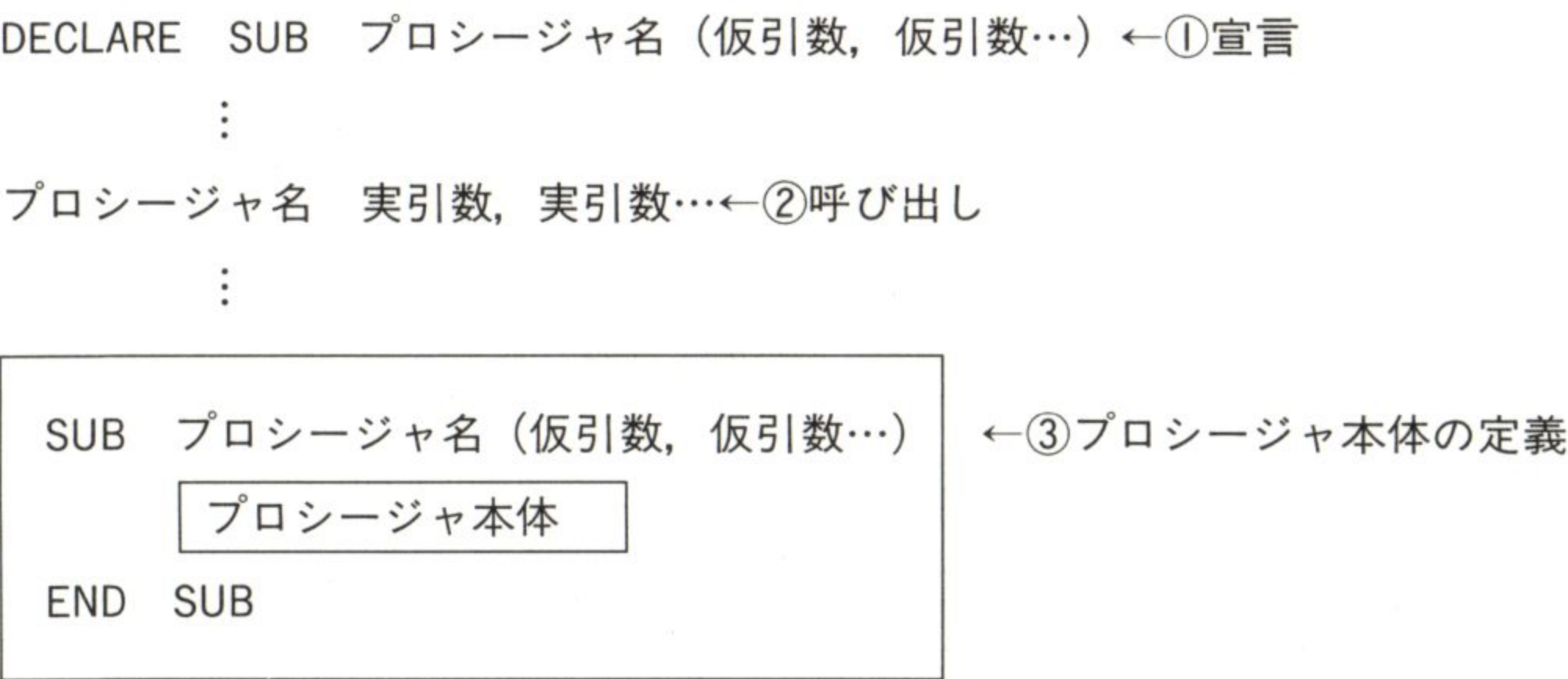
Quick BASICでは，従来型BASICの欠点を改良したサブルーチンをサポートします．この新しい形態のサブルーチンをプロシージャ(手続き)と呼びます．プロシージャは次の特徴を持ちます．

- プロシージャ間のデータ授受は引数(ひきすう)を用いて行うため，汎用的なデータ授受が行える．
- 各プロシージャ内の変数はそれぞれ独立(これをローカル変数という)なので，同じ名前の変数を使っても混同されない．
- ほかのモジュールから使用することができる．

Quick BASICのプロシージャは，SUB～END SUB(サブルーチン・プロシージャ)，FUNCTION～END FUNCTION(関数プロシージャ)の2種類で実現されます．

2 サブルーチン・プロシージャ

サブルーチン・プロシージャの定義方法，呼び出し方法を以下に示します．

```
DECLARE　SUB　プロシージャ名（仮引数，仮引数…）←①宣言
　　　　　：
プロシージャ名　実引数，実引数…←②呼び出し
　　　　　：
┌────────────────────────────┐
│ SUB　プロシージャ名（仮引数，仮引数…）│ ←③プロシージャ本体の定義
│　　　プロシージャ本体                 │
│ END　SUB                             │
└────────────────────────────┘
```

①プロシージャの宣言

メインルーチンにおいて，プロシージャの呼び出しを可能とするために，プロシージャの名前と引数を宣言します．

もし，この DECLARE 文をプログラマが記述しなければ，プログラムをセーブしたときに QB のエディタがプロシージャの定義部を参照して自動的に生成します．

②プロシージャの呼び出し

DECLARE でプロシージャを宣言しているので，プロシージャ名を書くだけで(CALL 命令を使わずに)プロシージャを呼び出すことができます．プロシージャ名の後に，プロシージャに渡すデータをカンマ(,)で区切って書きます．これを実引数(じつひきすう)といいます．

③サブルーチン・プロシージャの定義

サブルーチン・プロシージャは SUB と END SUB のあいだに定義します．SUB の後にプロシージャ名を書き，その後にデータを受けとる変数名をカンマ(,)で区切って書きます．これを仮引数(かりひきすう)といいます．

仮引数には型宣言をすることもできますが，省略すると暗黙の型宣言とみなされます．

3 関数プロシージャ

サブルーチン・プロシージャと関数プロシージャの大きな違いは，前者が戻り値を持たないのに対し，後者は戻り値を持つことです．

関数プロシージャの定義方法，呼び出し方法を以下に示します．

```
DECLARE  FUNCTION  プロシージャ名（仮引数，仮引数…）←①宣言
            ⋮
A＝プロシージャ名 （実引数，実引数，実引数…）←②呼び出し
            ⋮
┌───────────────────────────────────────────┐
│ FUNCTION  プロシージャ名（仮引数，仮引数…） │ ←③プロシージャ本体の定義
│        ┌──────────────────┐              │
│        │プロシージャ名＝値│              │
│        └──────────────────┘              │
│ END  FUNCTION                             │
└───────────────────────────────────────────┘
```

①関数プロシージャの宣言

メインルーチンにおいてプロシージャの呼び出しを可能とするために，プロシージャの名前と引数を宣言します．

なお，このDECLARE文をもしプログラマが記述しなければ，プログラムをセーブしたときにQBのエディタがプロシージャの定義部を参照して自動的に生成します．

②関数プロシージャの呼び出し

関数プロシージャの呼び出しは，「関数プロシージャ名(引数，…)」で行います．関数プロシージャの呼び出しは単独の文ではなく，式の中の一部として記述します．SIN( )，COS( )などの関数の扱いと同じです．

③関数プロシージャの定義

関数プロシージャの定義は，FUNCTIONとEND FUNCTIONの間にします．FUNCTIONの後に関数プロシージャ名を書き，その後にデータを受けとる引数をカンマ(,)で区切って書きます．

仮引数には型宣言をすることもできますが，省略すると暗黙の型宣言とみなされます．

関数プロシージャは値を呼び出し元に返すため，必ず一度，

関数プロシージャ名＝□

という代入文を行わなければなりません．この□の値が，関数プロシージャの値(戻り値)として呼び出し元に返されます．

■サブルーチン・プロシージャと関数プロシージャの違い

サブルーチン・プロシージャと関数プロシージャの違いを以下の表にまとめました．この点以外は，サブルーチン・プロシージャも関数プロシージャもまったく同じです．

●サブルーチン・プロシージャと関数プロシージャの違い

	サブルーチン・プロシージャ	関数プロシージャ
呼び出し方	1つの文として行う 実引数は(　)で囲まない Disp　"Hello",5,3	式の中の一部として行う 実引数は(　)で囲む Heikin＝Sum(a(　),10)/10
戻り値	サブルーチン・プロシージャ自体は値を持たない	関数プロシージャ自体が値を持つ．これを戻り値という．つまりSum(a(　),10)が合計の結果の値を持っていると考えられる

4 プロシージャの留意事項

QBでは各プロシージャをそれぞれ別のウィンドウで管理しているため，メインルーチンとプロシージャを同じウィンドウ内では見られません．各プロシージャおよびメインルーチンは，SHIFT＋F･2により順次切り換えて画面に表示できます．

また，QBのエディタ上で「SUBプロシージャ名(引数)↵」と入力すると，自動的にプロシージャを記述するウィンドウに切り換わります．

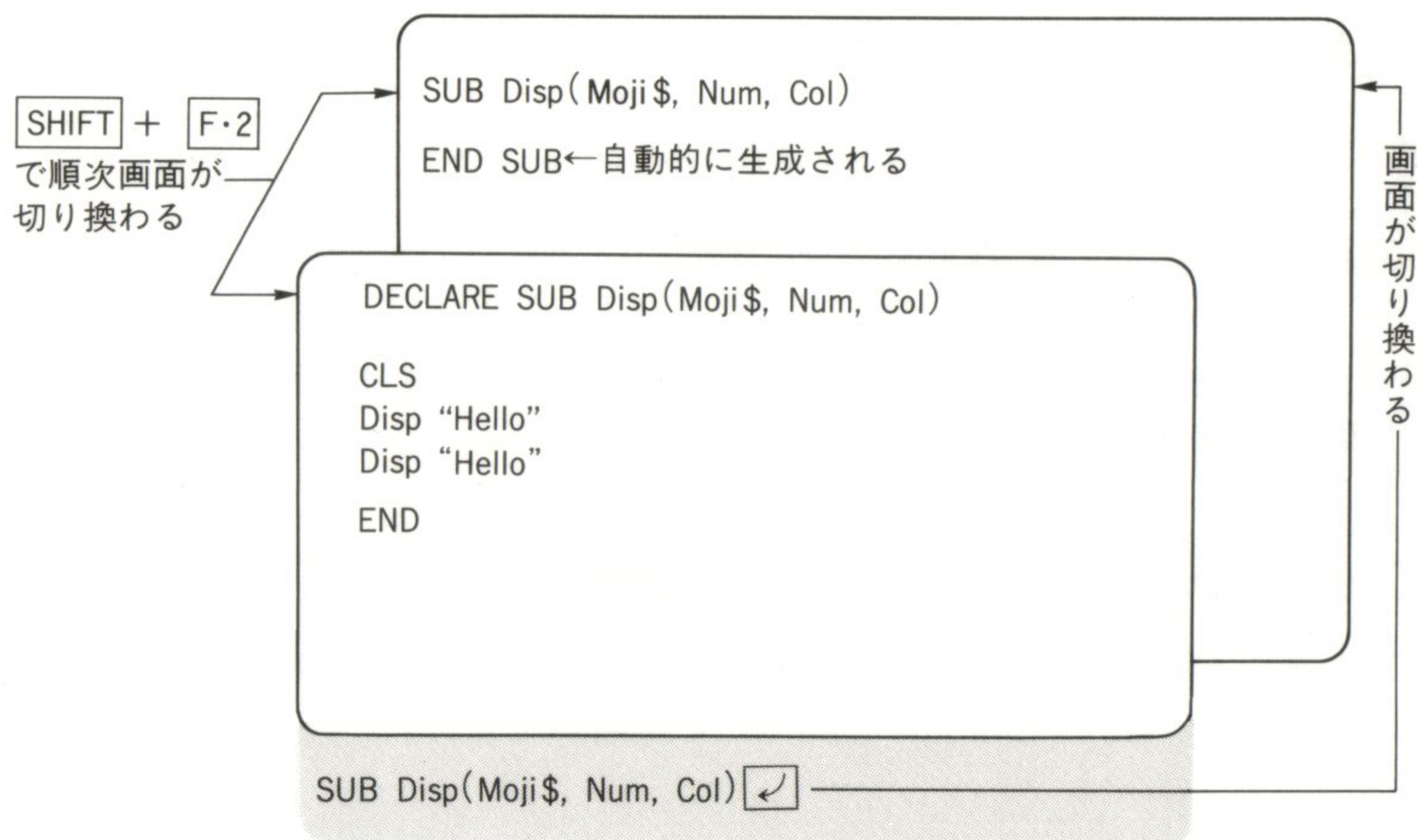
プロシージャのQB上での操作方法は第2章2-4の5を参照してください．

SHIFT ＋ F・2 で順次画面が切り換わる

```
SUB Disp(Moji$, Num, Col)
END SUB←自動的に生成される
```

```
DECLARE SUB Disp(Moji$, Num, Col)

CLS
Disp "Hello"
Disp "Hello"
END
```

SUB Disp(Moji$, Num, Col) ↵

画面が切り換わる

プロシージャ定義内では以下のステートメントを使用できません.

DEF FN ～END DEF, FUNCTION～END FUNCTION, SUB～END SUB
COMMON
DECLARE
DIM SHARED
OPTION BASE
TYPE～END TYPE

5 引数渡し

従来のサブルーチンは引数という概念を持たないため，サブルーチン内で使用している変数にメイン側でデータを代入してから，サブルーチンコールするという煩雑な処理になってしまいます.

ところが，プロシージャを用いれば引数を用いて次のように簡単に，しかも汎用的にデータを渡すことができます.

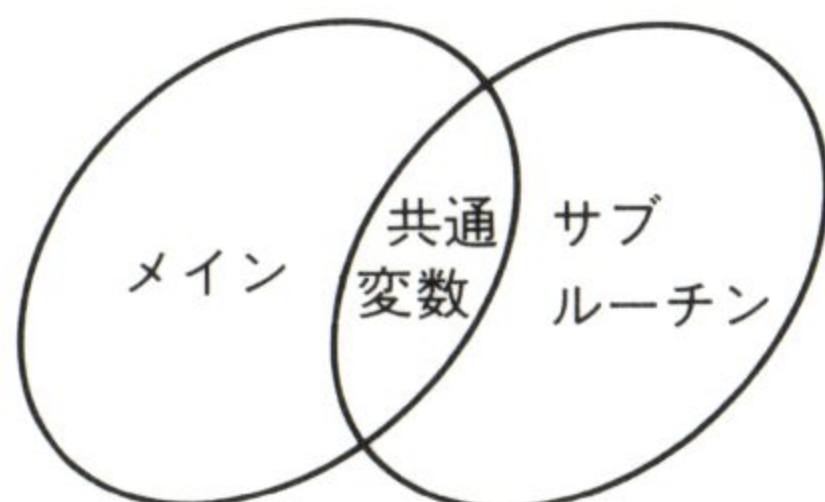
```
Disp  "Hello Quick BASIC", 2, 3
 :         ↓                       引数渡し
SUB  Disp(Moji$, Num, Col)
 :                          ←─プロシージャ
END  SUB
```

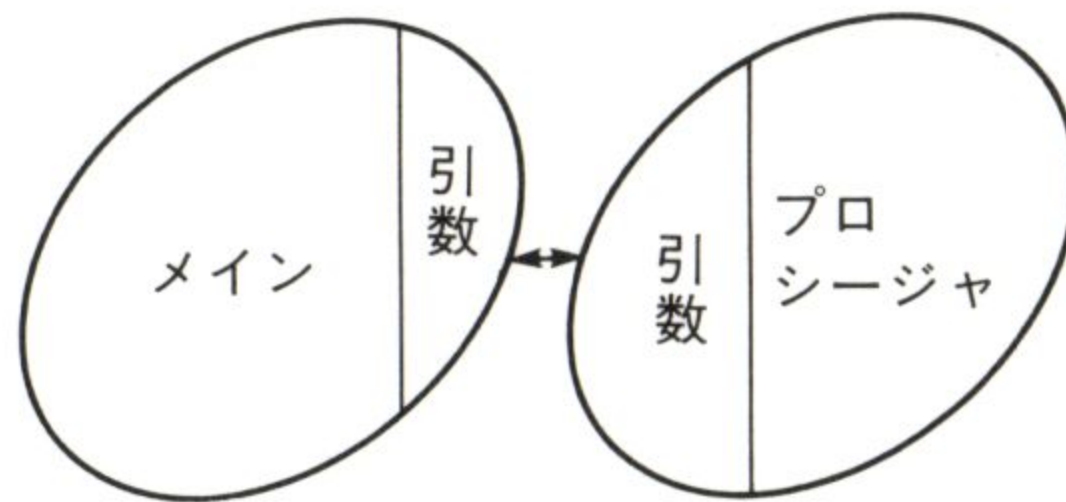
プロシージャの呼び出し時に書く引数は，プロシージャに渡す実際のデータなので実引数(じつひきすう)と呼び，変数，定数，式が書けます．

これに対し，プロシージャの定義側で書く引数を仮引数(かりひきすう)と呼びます．仮引数はデータを受けとるものなので変数しか指定できません．

実引数と仮引数の名前は違っていてもよいですが，データの型は一致していなければなりません．

●従来型サブルーチン

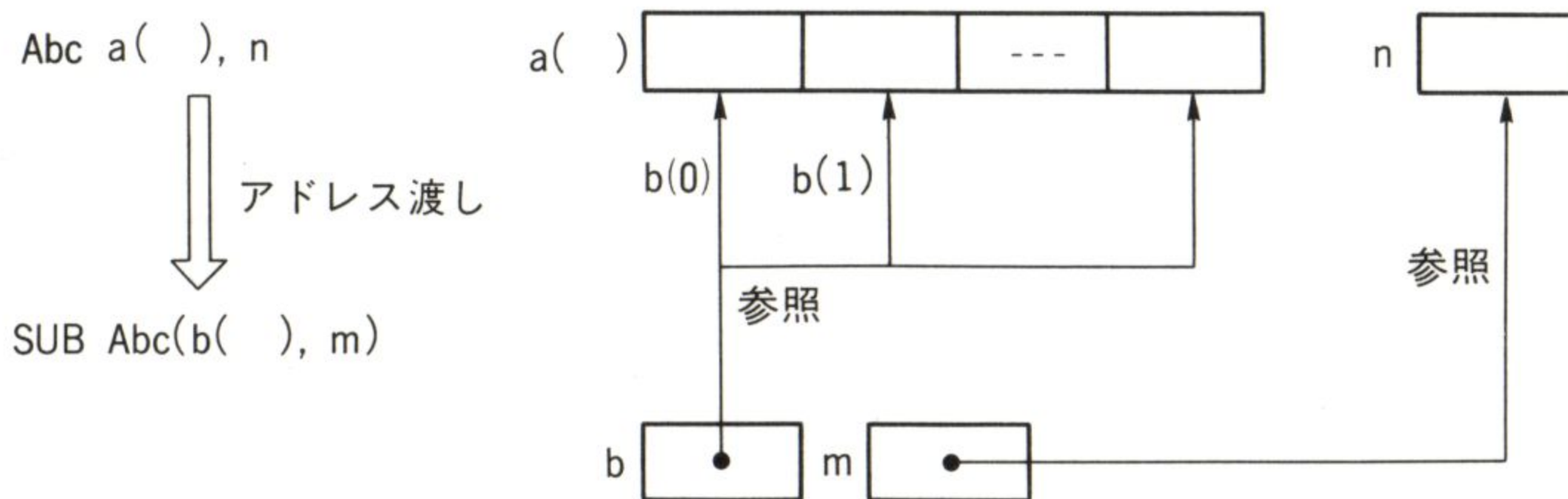
メイン
共通
変数
サブ
ルーチン

●プロシージャ

メイン
引
数
引
数
プロ
シージャ

Quick BASICの引数渡しでは，参照による呼び出し(call by reference)と，値による呼び出し(call by value)という2つの方法が行えます．

■参照による呼び出し (call by reference)

Quick BASICのプロシージャ間の引数渡しは，原則的には参照による呼び出しで行われます．

Abc a(　), n
アドレス渡し
SUB Abc(b(　), m)

a(　)
---
n
b(0)
b(1)
参照
参照
b
m

参照による呼び出しでは，実引数のアドレスが仮引数に渡され，プロシージャ側ではこのアドレスをもとに，呼び出し元の仮引数を参照します．したがって，プロシージャ側で引数の値を変更すると，呼び出し側の引数の値も変化することになります．

つまり，仮引数は実引数と連動していて，プロシージャ側で仮引数の値を変えればその結果は実引数にも影響をおよぼします．

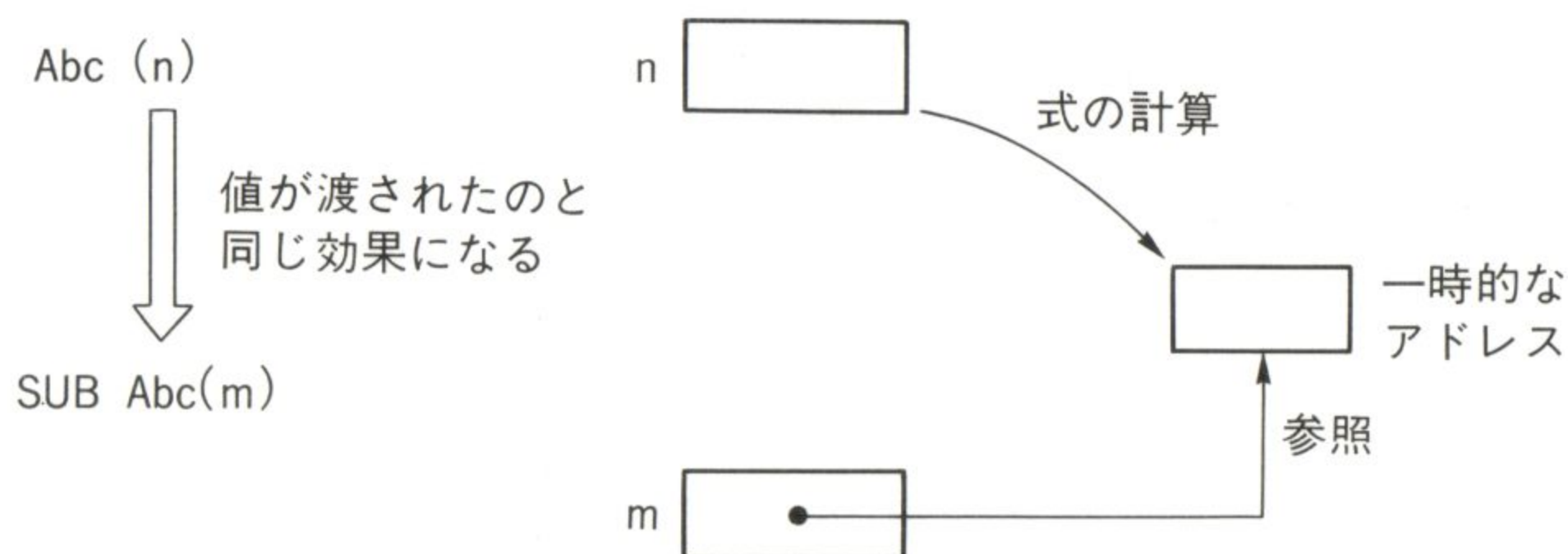
```
a$= "Quick"
n = 3
Disp  a$,  n,  5
  :
SUB  Disp (Moji$,  Num,  Col)
       Moji$= "Turbo"
       Num= 7

END  SUB
```

プロシージャで引数 Moji$，Num の値を変えていますが，これは同時にメインルーチンの引数 A$, n の値を変えていることになるのです．したがって，プロシージャから戻ったときの a$，n の値はそれぞれ“Turbo”, 7 になっています．

これはあたかも，Moji$，Num を介してメイン側の a$，n に値を返しているようなものです．

■値による呼び出し（call by value）

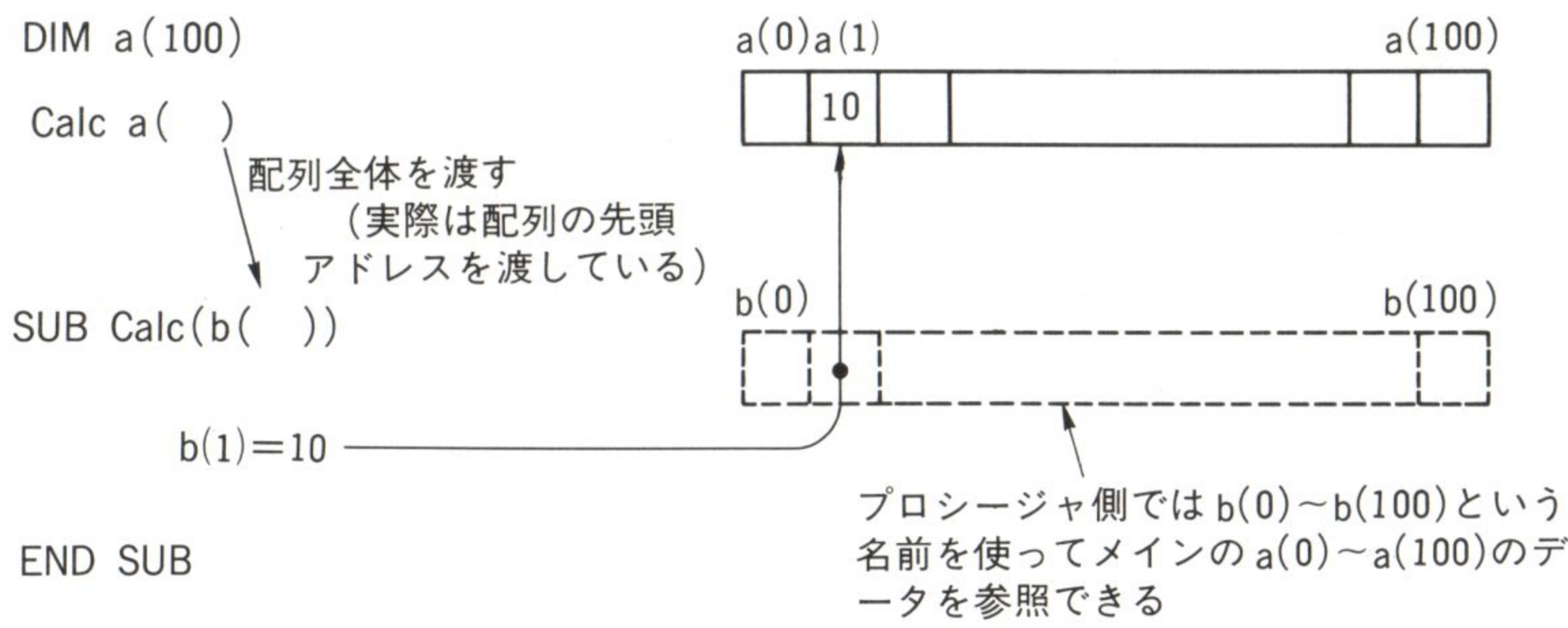
Quick BASIC の引数渡しは，あくまでも参照による呼び出しで行っていますが，仮引数を式として与えることにより，値による呼び出しと同じ機能を実現できます．式とはn＋1，5，(n)などです．(　)をつければ，式にできることに注意してください．

Abc (n)
値が渡されたのと
同じ効果になる
SUB Abc(m)
n
式の計算
一時的な
アドレス
参照
m

仮引数を式とすることで，その値が一時的なアドレスに記憶され，そのアドレスが実引数に渡されることになります．したがって，プロシージャ側でmの値を変えても，呼び出し側のnの値は変化を受けません．

値による呼び出しは，再帰プロシージャを記述する際に使います．

■配列データを渡す

配列データの全体をプロシージャに渡すには，引数として配列名の後に添字のない(　)を付けた「配列名(　)」を用います．

```
DIM a(100)

Calc a( )

SUB Calc(b( ))

b(1)=10

END SUB
```


プロシージャ側では，b(0)~b(100)という配列を使ってメイン側のa(0)~a(100)という配列を操作することができます．また，プロシージャ側でb(1)=10などとすると，メイン側のa(1)も10に変わります．

このように，メイン側のa(0)~a(100)とプロシージャ側のb(0)~b(100)は連動していて，プロシージャ側でb(0)~b(100)を変化させると，メイン側のa(0)~a(100)も変化します．このことを利用して，プロシージャ側からメイン側に配列データを返すことができます．

なお，プロシージャ側では，渡された配列について，その先頭アドレスしかわかっていません．配列のサイズを調べたいときは，LBOUND / UBOUND関数を用いてください．

■レコード・データを渡す

レコード・データは，全体でもメンバ単位でもプロシージャに渡すことができます．

```
TYPE   Eisei
   Star   AS   STRING *10
   Kodo   AS   INTEGER
   Shuki  AS   SINGLE
END  TYPE

DIM   a   AS   Eisei
 :
 Disp   a
 :
SUB   Disp (b   AS   Eisei)
 :
END  SUB
```

6 DEF FN 関数

DEF FN 関数は，関数プロシージャと似ていますが，プロシージャではありません．DEF FN 関数が定義されているモジュール以外から呼び出すことはできません．

DEF FN 関数は次のように定義します．

```
DEF  FNMax(a, b)
   ⋮
  FNMax=□
END  DEF
```

関数名は FN で始めます．DEF FN 関数の引数渡しは，値による呼び出し(call by value)で行いますから，関数側から呼び出し元に引数を介してデータを返すことはできません．

DEF FN 関数は再帰的に使うことはできません．

7 再帰

Quick BASIC では，再帰呼び出し(recursive call)が行えます．再帰呼び出しは，プロシージャの内部から自分自身のプロシージャを呼び出すコール方法です．

再帰呼び出しは，一般に次のような形式をとります．

```
   ⋮
rproc [実引数, …]  ←── 再帰プロシージャの最初の呼び出し
   ⋮
SUB  rproc ([仮引数, …])←── 再帰プロシージャの定義
   IF   脱出条件  THEN  EXIT  SUB  ←── 再帰呼び出しからの脱出

   rproc  [実引数, …]  ←── 再帰呼び出し

END  SUB
```

例

ハノイの塔問題の再帰解をQBでプログラムしたものです.

```
' ---/* ハノイの塔問題の再帰解 */---

DECLARE SUB Hanoi (n%, a$, b$, c$)

INPUT "円盤の枚数 "; n%
Hanoi n%, "a", "b" ,"c"
END


SUB Hanoi (n%, a$, b$, c$)
    IF (n% > 0) THEN
        Hanoi n% - 1, a$, c$, b$
        PRINT n%; "番の板を "; a$; " から  "; b$: " に移動"
        Hanoi n% - 1, c$, b$, a$
    END IF
END SUB
```

```
円盤の枚数 ？4
 1番の板を a から c に移動
 2番の板を a から b に移動
 1番の板を c から b に移動
 3番の板を a から c に移動
 1番の板を b から a に移動
 2番の板を b から c に移動
 1番の板を a から c に移動
 4番の板を a から b に移動
 1番の板を c から b に移動
 2番の板を c から a に移動
 1番の板を b から a に移動
 3番の板を c から b に移動
 1番の板を a から c に移動
 2番の板を a から b に移動
 1番の板を c から b に移動
```

4-6 メタコマンド

1 メタコマンドとは

メタコマンドは，コメントステートメント('またはREM)の後に次のように書きます．

'$メタコマンド

メタコマンドは，プログラムの実行に先立って，Quick BASICに対し，ある処理を指示するもので，$INCLUDE，$DYNAMIC，$STATICの3つがあります．

2 $INCLUDE

$INCLUDEは，別のプログラム・ファイルを取り込むもので次のように書きます．

'$INCLUDE：'ファイル名'

これで，この位置に〈ファイル名〉で指定されるファイルを取り込みます．

インクルード・ファイルはテキストファイルでなければなりません．インクルード・ファイルの作り方は，第2章2-4の7を参照してください．

インクルード・ファイルにはプロシージャを記述できません．

3 $DYNAMIC，$STATIC

$DYNAMICと$STATICは，配列の記憶領域の割り当て方を指示するもので，前者は動的配列，後者は静的配列の割り当てを指示します．

```
'$DYNAMIC
DIM  a(100)   ←  動的配列
'$STATIC
DIM  b(100) ┐
DIM  c(100) ┘ ←  静的配列
```

第5章

ステートメントと関数の概要

5-1 実行制御

プログラムの流れを制御するステートメントで，プログラムを作る上で基本になるものです．詳しくは第 4 章 4-4を参照してください．

ステートメント	機　　能
CHAIN	別のプログラムに実行を移す
DO～LOOP	前判定/後判定反復
EXIT	ループ，プロシージャからの脱出
FOR TO STEP～NEXT	所定回反復
GO TO	分岐
GOSUB	サブルーチンの呼び出し
IF GOTO ELSE	単純 IF
IF THEN ELSE	単純 IF
IF THEN～ELSEIF～END IF	ブロック IF
ON GOSUB	条件によるサブルーチン・コール
ON GOTO	条件による分岐
RETURN	サブルーチンからのリターン
SELECT CASE	複数条件判断
STOP	実行の停止
WHILE～WEND	前判定反復

5-2 サブモジュール

プロシージャの宣言，定義，呼び出しを行うステートメントです．詳しくは第4章4-5を参照してください．

ステートメント	機　　能
CALL	プロシージャの呼び出し
CALLS	引数をfarアドレスとして渡すプロシージャ・コール
DECLARE SUB	サブルーチン・プロシージャの宣言
DECLARE FUNCTION	関数プロシージャの宣言
DEF FN	関数の定義（1行）
DEF FN～END DEF	関数の定義（複数行）
FUNCTION～END FUNCTION	関数プロシージャの定義
SUB～END SUB	サブルーチン・プロシージャの定義

5−3 変数管理

変数の型宣言，レコード型の定義，変数のスコープ指定，変数の初期化，変数の割り当てられているアドレスの取得などを行うステートメントと関数です．詳しくは第4章4-2を参照してください．

	ステートメント/関数	機　　能
宣言と定義	COMMON	共用変数の宣言
	CONST	記号定数の定義
	DIM	変数または配列の型宣言
	DEF INT/SNG/DBL /LNG/STR	変数のインプリシット宣言
	SHARED	プロシージャ間の変数の共有
	STATIC	静的変数の宣言
	TYPE～END TYPE	レコード型の定義
初期化	CLEAR	変数の初期化
	ERASE	配列変数の初期化と解放
	REDIM	動的配列の割り当てスペースの変更
変数のアドレス	SADD　（関数）	文字列が格納されているアドレスの取得
	VARPTR　（ 〃 ）	変数のアドレスのオフセット値の取得
	VARPTR$　（ 〃 ）	変数のアドレスを文字列として取得
	VARSEG　（ 〃 ）	変数のアドレスのセグメント値の取得
添字	OPTION BASE	配列添字の下限デフォルト値の設定
	LBOUND　（関数）	配列添字の下限値の取得
	UBOUND　（ 〃 ）	配列添字の上限値の取得
	LEN　（ 〃 ）	変数のサイズの取得

セグメント値とオフセット値については，本章5-13を参照してください．

5-4 ファイル処理

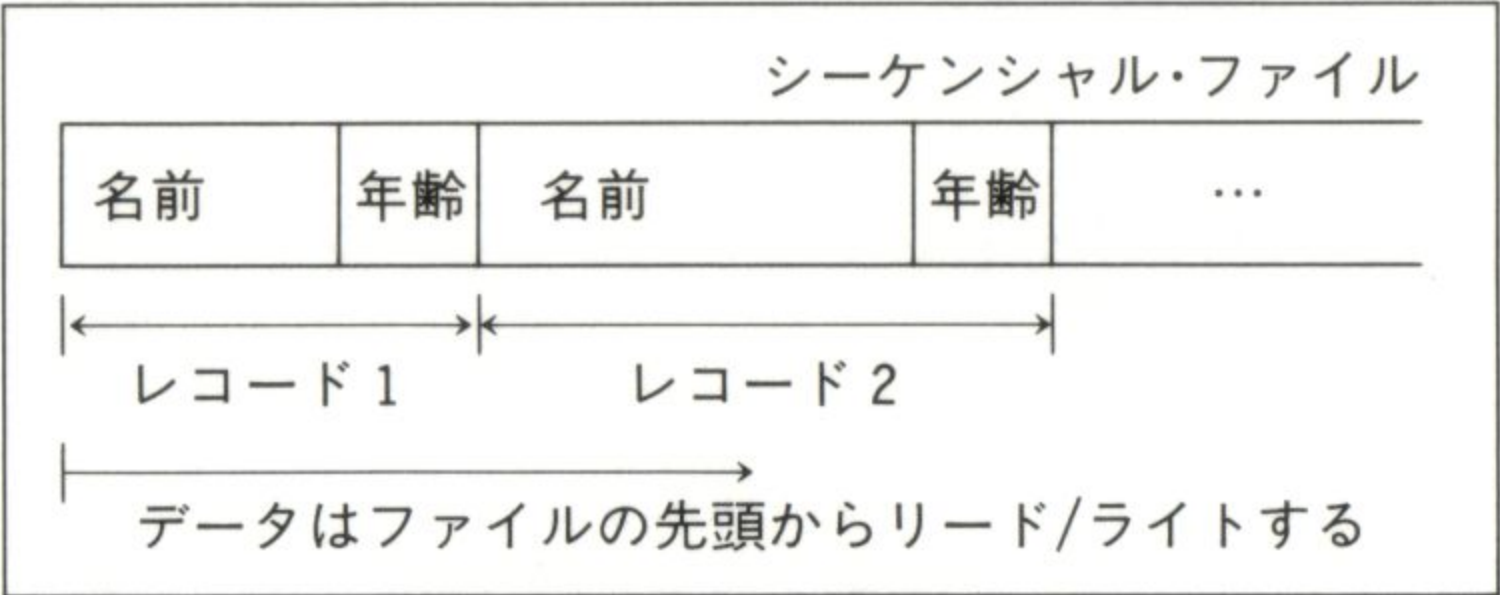
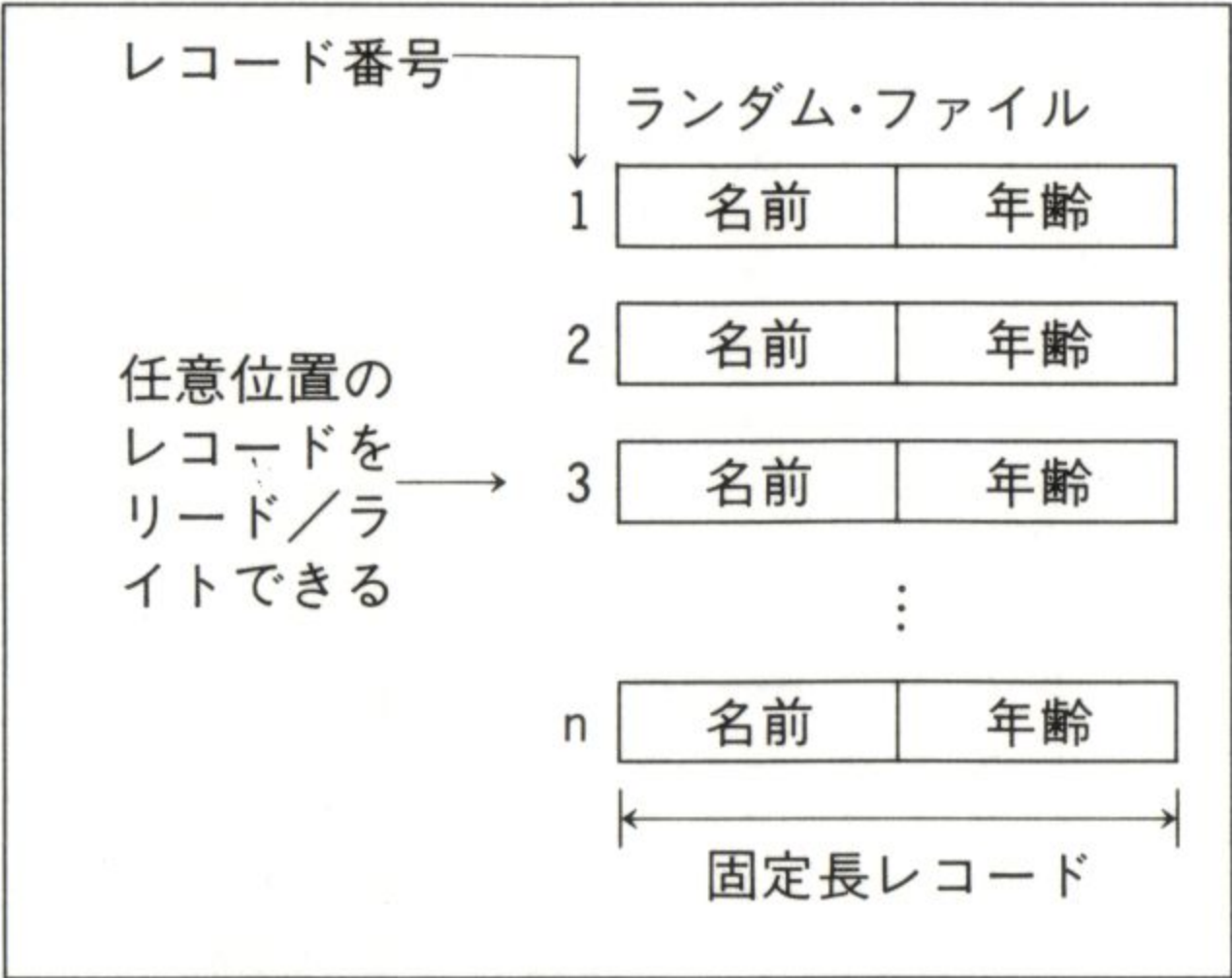
ファイルのオープン/クローズ，ファイルとのデータの入出力，ファイル情報の取得を行うステートメントと関数です.

	ステートメント/関数		機　能
オープン	OPEN		ファイルのオープン
	CLOSE		ファイルのクローズ
	LOCK		他のプロセスからのアクセスの禁止
	UNLOCK		他のプロセスからのアクセスの禁止の解除
シーケンシャル・アクセス	INPUT$	(関数)	指定したバイト数分の文字列のリード
	INPUT¥	(〃)	指定した長さの文字のリード
	INPUT #		シーケンシャル・ファイルからのリード
	LINE INPUT #		シーケンシャル・ファイルから1行単位でリード
	PRINT #		シーケンシャル・ファイルへのライト
	WRITE #		シーケンシャル・ファイルへのライト
ランダム・アクセス	FIELD		ファイルバッファの設定
	LSET		ファイルバッファへの値の設定
	RSET		ファイルバッファへの値の設定
	GET #		ランダム・ファイルからのリード
	PUT #		ランダム・ファイルへのライト
ファイル情報	EOF	(関数)	ファイル終わりの検知
	FILEATTR	(〃)	ファイル属性の取得
	FREEFILE	(〃)	未使用ファイル番号の取得
	LOC	(〃)	ファイル現在位置の取得
	LOF	(〃)	ファイルサイズの取得
	SEEK	(〃)	ファイル現在位置の取得
	SEEK #		ファイル現在位置の移動

■ファイルの種類

BASIC で扱えるファイルは，シーケンシャル・ファイルとランダム・ファイルの 2 つです．

シーケンシャル・ファイルは，データをファイルの先頭から順次リード/ライトしていくものです．

1 件のレコードの長さは可変長であってもかまいません．

これに対しランダム・ファイルは，任意のレコード位置のデータをリード/ライトすることができます．1 件のレコードの長さは，固定長でなければなりません．

■ファイルのオープンとファイル番号

ファイルのオープンにより，ファイル名で示されるファイルを，指定したファイル番号(#n)で処理できるようになります．

```
                                  ⎧INPUT ⎫
OPEN  "d:filename.typ" FOR ⎨OUTPUT⎬ AS #n
                                  ⎩APPEND⎭     ↑
                                               ├ファイル番号
                                               ↓
OPEN  "d:filename.typ" FOR  RANDOM  AS #n  LEN= レコード長
```

オープンモードの意味は下表のとおりです．

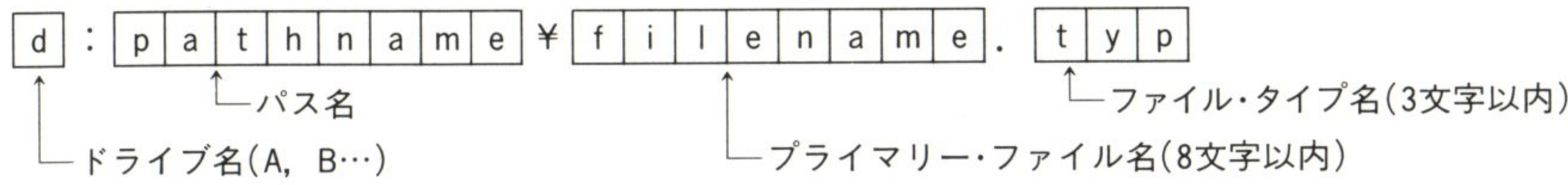
モード	意　味
INPUT OUTPUT APPEND	すでに作られているファイルからデータを読む 新規にファイルを作成し，データを書く すでにあるファイルの後にデータを追加書き込み
RANDOM	ランダム・ファイルとしてリード/ライトモードでオープン
BINARY	バイナリモードでオープン

ファイルへのデータ処理が終了したら，

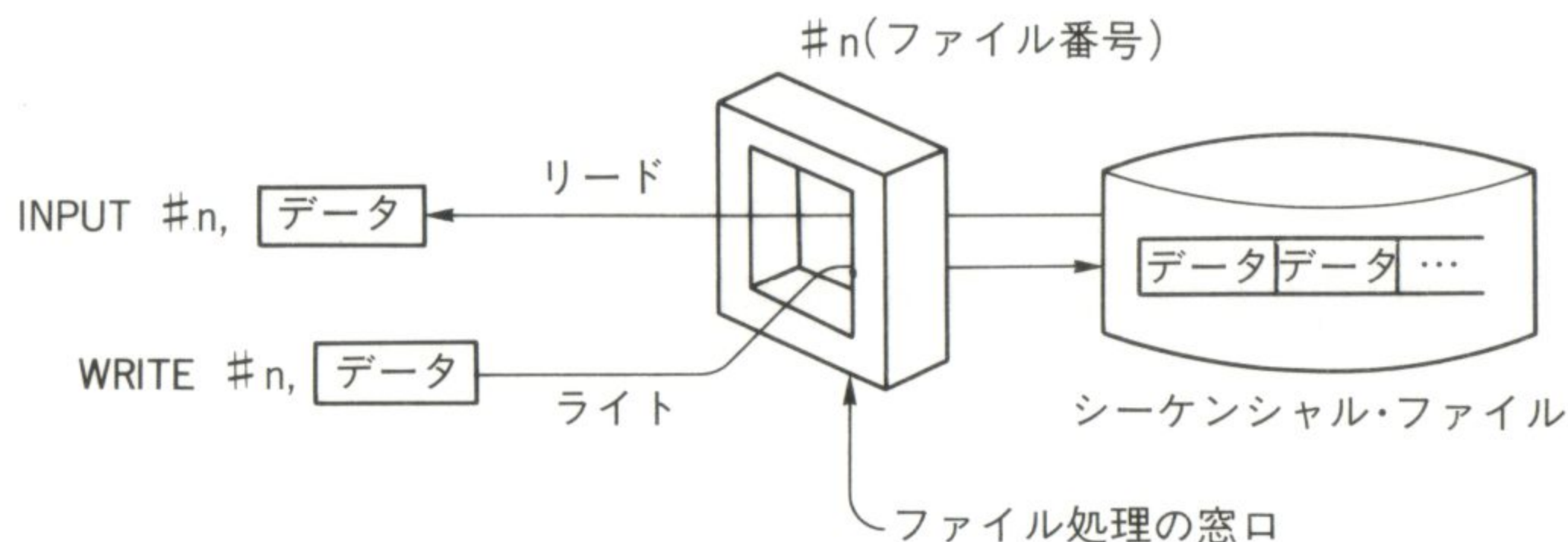
```
CLOSE  #n
```

により，ファイルをクローズしてファイル番号を解放します．

QBで扱うファイル名は，以下に示すMS-DOSのファイル名の規則にしたがいます．

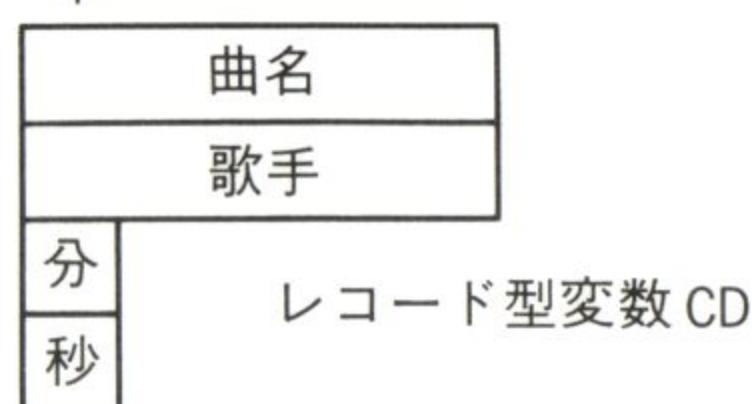
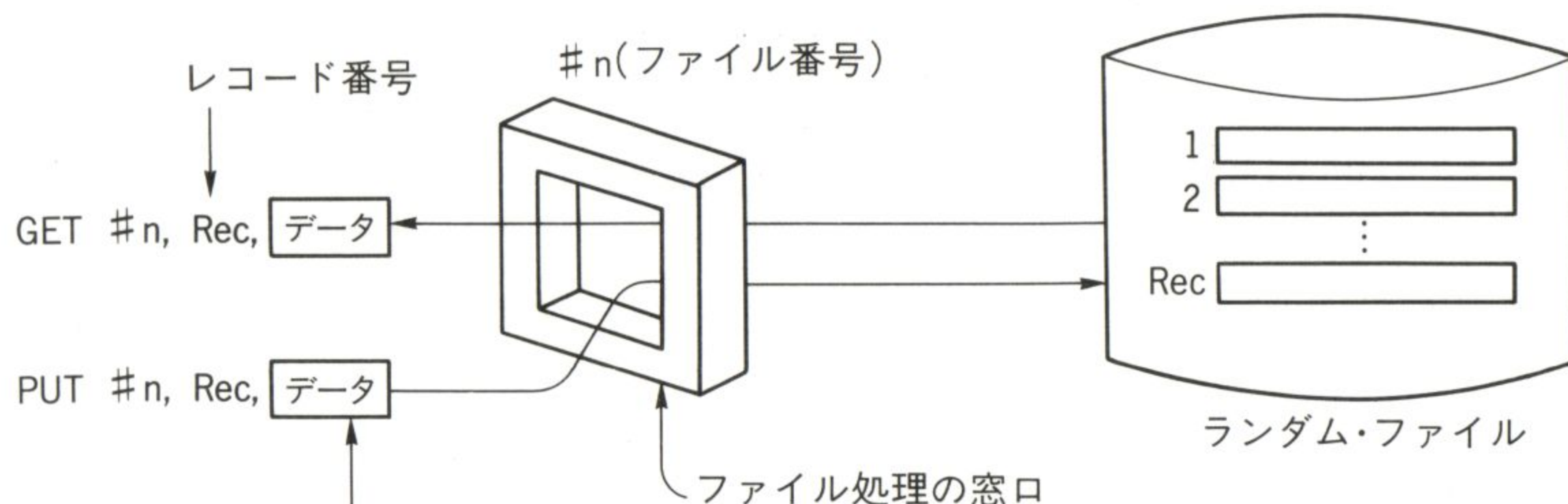
```
d : pathname ¥ filename . typ
↑    ↑           ↑           ↑
|    └パス名      |           └ファイル・タイプ名(3文字以内)
|                └プライマリー・ファイル名(8文字以内)
└ドライブ名(A, B…)
```

なお，ファイル名の英字の大／小文字は区別されず同じものとして扱われます．つまりtest.datとTEST.DATは同じファイル名となります．

■シーケンシャル・ファイルのリード/ライト

ファイル・オープンされているファイルには，ファイル番号を通してデータをリード/ライトします．

ファイルをリードしていったとき，ファイルのエンドはEOF 関数で調べます．EOF(n)はファイル番号nのファイルがファイル・エンドなら真(−1)を返します．したがって，ファイル・エンドまでファイルからデータをリードするには，次のように書きます．

```
WHILE NOT EOF(n)
   INPUT #n, [    ]
      :
WEND
```

■ランダム・ファイルのリード/ライト

ランダム・ファイルは，指定したレコード番号のデータを即座にリード/ライトすることができます．


```
TYPE CDMusic
  Song   AS STRING*20
  Singer AS STRING*20
  Min    AS INTEGER
  Sec    AS INTEGER
END TYPE

DIM CD AS CDMusic
```

レコードの長さは,

LEN(レコード型変数名)

で求められ，ファイルのサイズは,

LOF(ファイル番号)

で求められるので，ランダム・ファイルのレコード数は,

LOF(ファイル番号) ¥ LEN(レコード型変数)

で求められます.

したがって，ランダム・ファイル(ファイル番号 n)のファイル・エンドの次にリード/ライト位置を移動するには,

SEEK n, LOF(n) ¥ LEN(レコード型変数名)+1

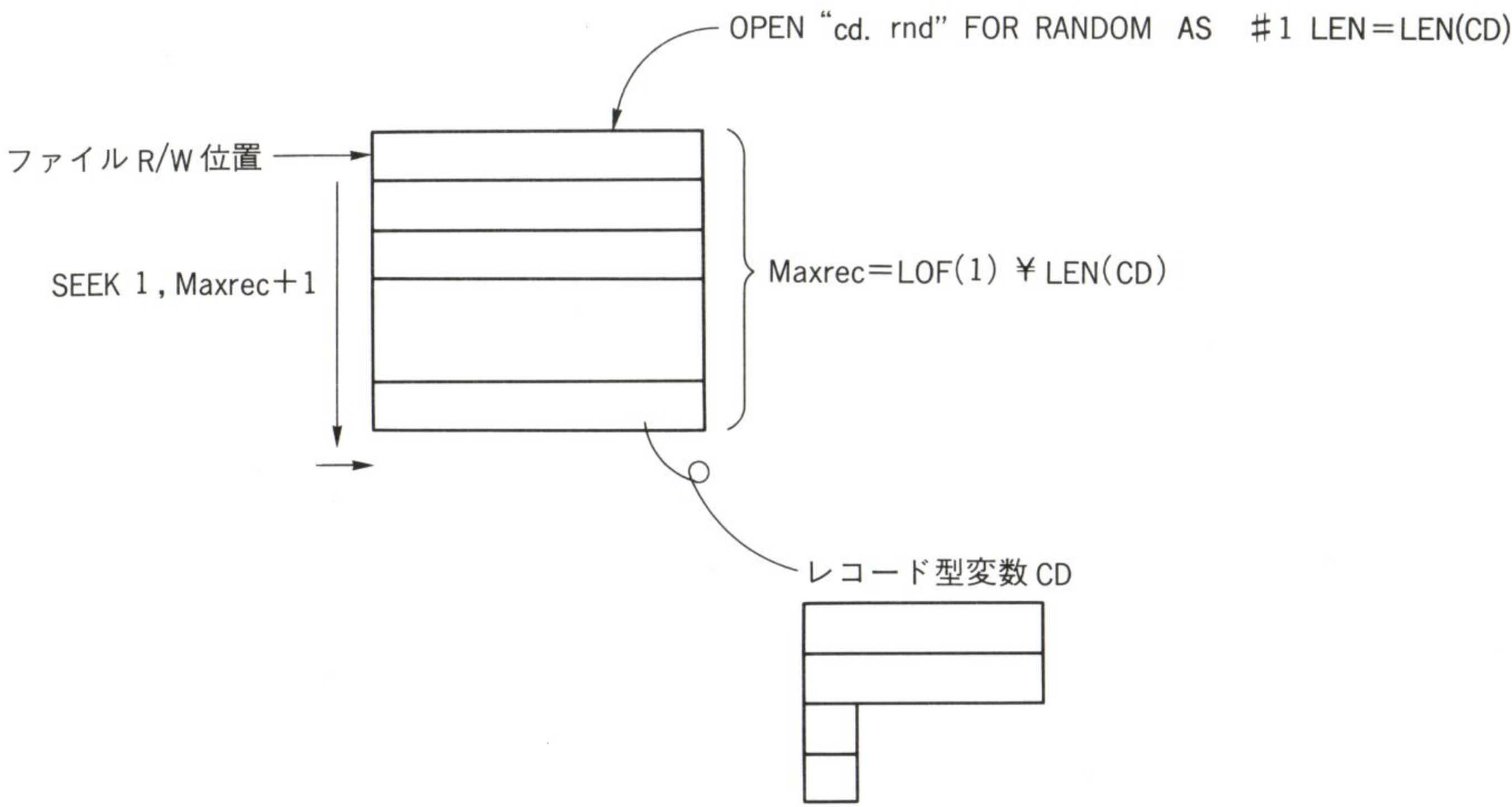
とします.

OPEN "cd. rnd" FOR RANDOM AS #1 LEN=LEN(CD)

ファイル R/W 位置

SEEK 1, Maxrec+1

Maxrec=LOF(1) ¥ LEN(CD)

レコード型変数 CD

■デバイス・ファイル

Quick BASICでは次のようなデバイスをOPENによりオープンすることで，ディスク・ファイルと同様にファイルとして扱うことができます．

デバイス名	デバイス	入出力モード
COM1：	シリアルポート1	入出力
COM2：	シリアルポート2	入出力
CONS：	スクリーン	出力
KYBD：	キーボード	入力
LPT1：	プリンタ1	出力
LPT2：	プリンタ2	出力
LPT3：	プリンタ3	出力
SCRN：	スクリーン	出力

たとえば，プリンタのオープンとそれへの出力は，次のように行います．

```
OPEN  "LPT1:" FOR OUTPUT AS #2        ←プリンタのオープン
  ⋮
PRINT #2, "HELLO"                     ←プリンタに"HELLO"を出力
```

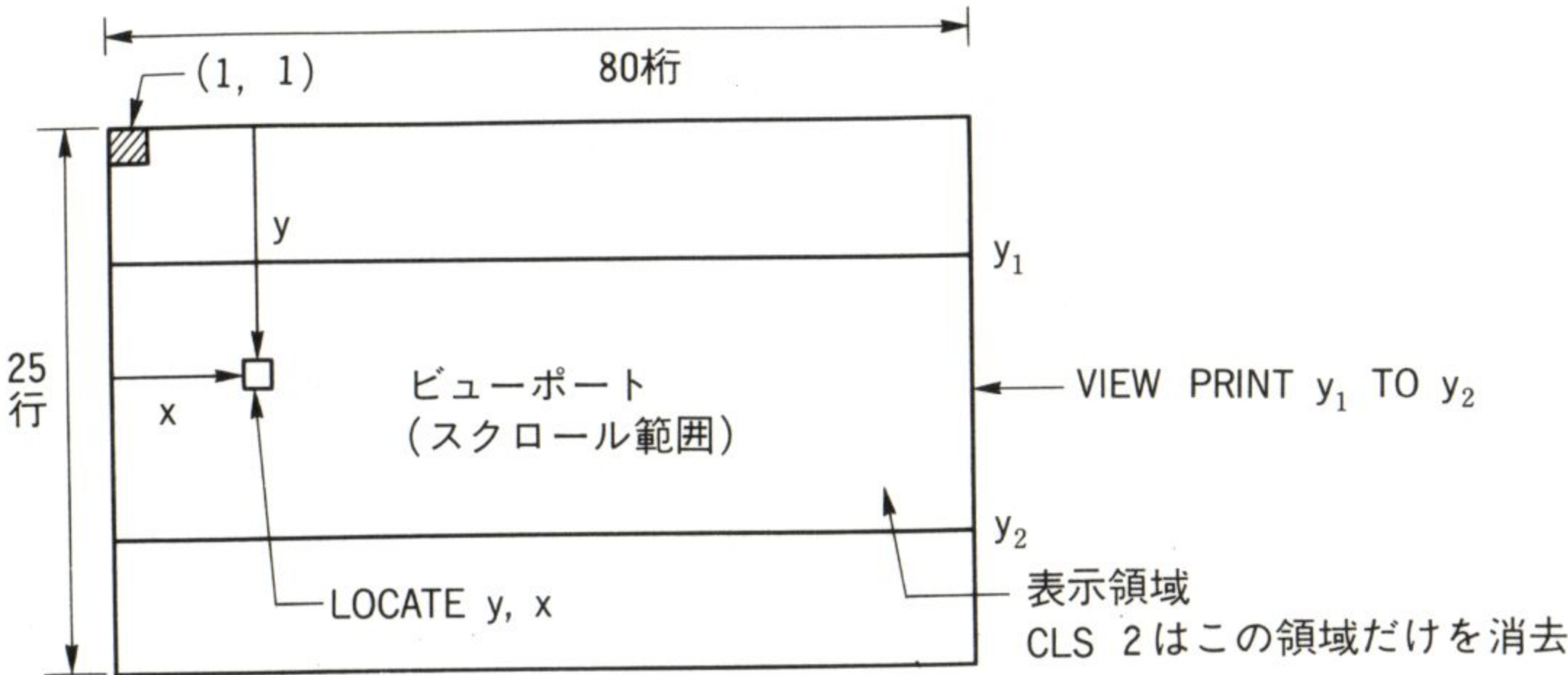
5-5 コンソール入出力

キーボードからの入力，スクリーン(CRT)への出力を行うステートメントと関数です．

	ステートメント/関数	機　　能
キーボード	INKEY$	待ったなしのキー入力
	INPUT	キーボードからのデータ入力
	KEY	ファンクションキー制御
	LINE INPUT	キーボードから1行入力
スクリーン	CSRLIN　(関数)	現在のカーソル行の取得
	LOCATE	カーソルを指定位置に移す
	POS　(関数)	現在のカーソル欄の取得
	PRINT	スクリーンへの出力
	PRINT USING	書式指定付きのスクリーンへの出力
	SCREEN　(関数)	指定位置の文字または色の取得
	SPC　(関数)	スペースの出力
	TAB　(関数)	表示位置の指定
	VIEW PRINT	テキストビューポートの上下行の指定
	WIDTH	画面幅の設定
	WRITE	スクリーンへの出力

■画面構成

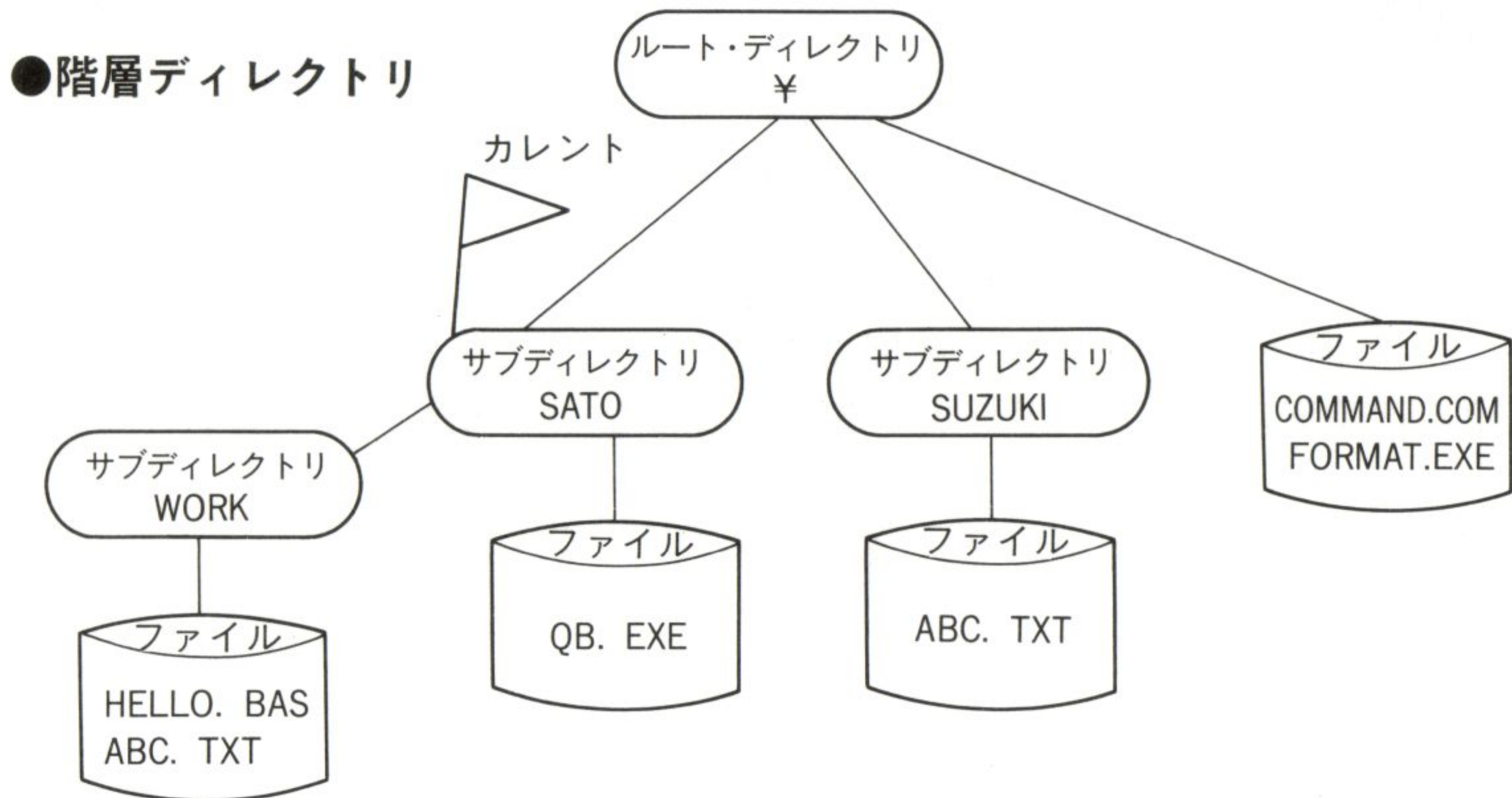
テキスト出力画面は80桁×25行のサイズで，左隅上の座標が(1，1)です．WIDTH 文がありますが，98用 Ver.4.2の画面サイズは80×25の固定です．Ver.4.5では，80×20もサポートしています．

VIEW PRINT 文により画面上への表示範囲(ビューポート)を設定することができます．CLS 2により消去される範囲はこのビューポートの中だけです．LOCATE で指示する行位置は，ビューポートの開始行ではなく，物理画面の最上行です．

5-6 ディスク・ファイル操作

ディスク上のファイルと階層ディレクトリの操作を行うステートメントです．

	ステートメント	機能
階層ディレクトリ	CHDIR MKDIR RMDIR	カレント・ディレクトリの変更 サブディレクトリの作成 サブディレクトリの削除
ファイル	FILES KILL NAME	ファイル一覧(ディレクトリ)の表示 ファイルの削除 ファイル名の変更

■階層ディレクトリ

1つのディスクを複数のユーザが共同利用する場合(特にハードディスク・システムでこの傾向が強い)や，ファイルの種類が増えてくると，単一のディレクトリ構造では効率的なファイル管理が行えません．そこで，図に示すような階層ディレクトリ構造が考えられています．

●階層ディレクトリ

階層ディレクトリでは，一番のもとになるディレクトリをルート・ディレクトリと呼び，その下にいくつかのサブディレクトリを持っています．現在の作業のもとになっているディレクトリをカレント・ディレクトリといいます．システム起動時はカレント・ディレクトリはルート・ディレクトリに設定されています．

5−7 デバイス操作

プリンタ，通信回線，入出力ポートなどに対する操作を行うステートメントと関数です．

	ステートメント/関数	機　　能
プリンタ	LPOS　（関数） LPRINT WIDTH LPRINT	印刷行バッファ内のプリンタ・ヘッドの現在位置の取得 プリンタへのデータ出力 プリンタの1行幅の設定
ポート入出力	OPEN COM INP　（関数） OUT WAIT	通信ポートのオープン ポートから1バイト入力 ポートへ1バイト出力 入力ポートの状態を調べる間プログラムを停止
デバイスドライバ	IOCTL$　（関数） IOCTL	デバイス・ドライバから制御文字列を取得 制御文字列をデバイス・ドライバに送る

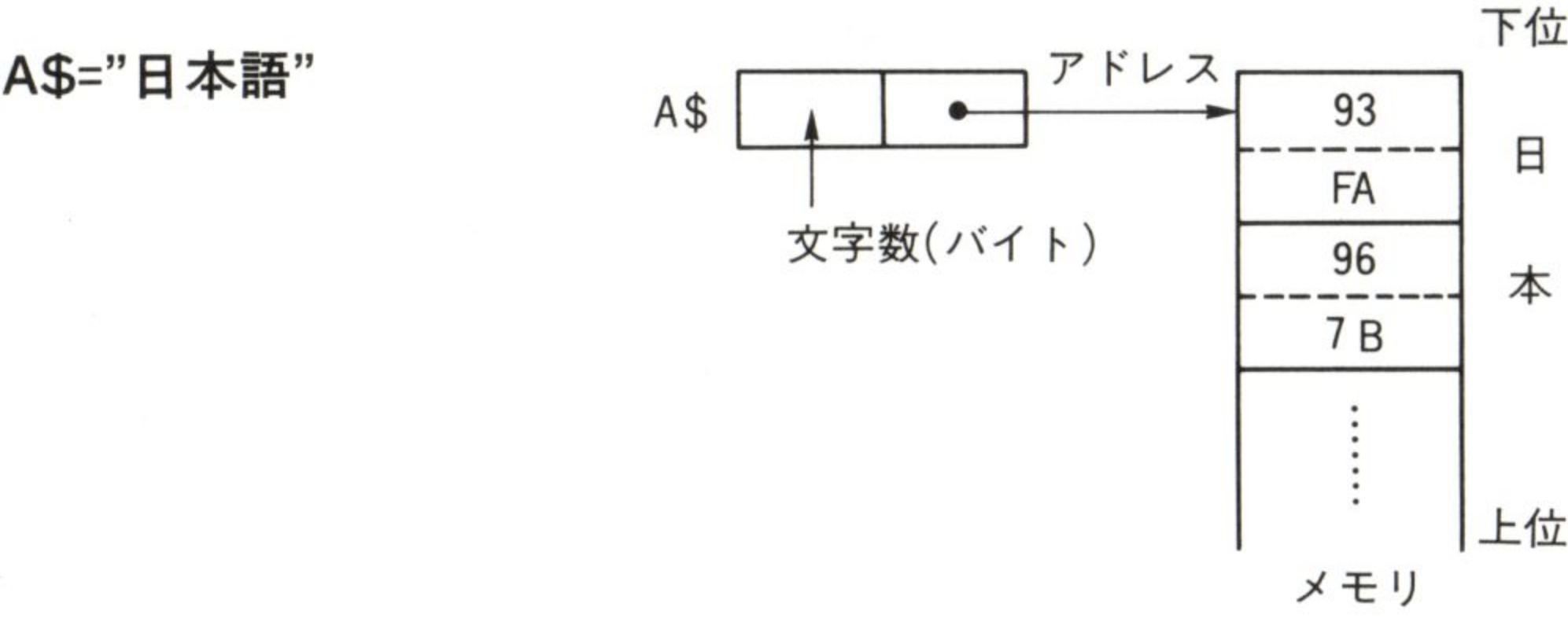
5-8 文字列処理

文字列に対する処理を行うためのステートメントと関数です.

	ステートメント/関数		機　能
一般文字列	INSTR		文字列の検索
	LEFT$	(関数)	左部分文字列の取得
	LEN	(〃)	文字列長さの取得
	MID$	(〃)	中間文字列の取得
	MID$		文字列の置き換え
	RIGHT$	(関数)	右部分文字列の取得
漢字	KINSTR		INSTR の漢字対応
	KLEN	(関数)	LEN の漢字対応
	KMID$	(〃)	中間文字列の取得
	KMID		文字列の置き換え
コード	ASC	(関数)	文字を ASCII コードに変換
	CHR$	(〃)	ASCII コードを文字に変換
	JIS$	(〃)	文字を JIS コードに変換
	KTN$	(〃)	文字を句点コードまたは ASCII コードに変換
変換	CDBL$	(関数)	1バイト文字を2バイト文字に変換
	CSNG$	(〃)	2バイト文字を1バイト文字に変換
	LCASE$	(〃)	小文字に変換
	UCASE$	(〃)	大文字に変換
その他	KEXT$	(関数)	1バイト文字または2バイト文字の抽出
	LTRIM$	(〃)	左側スペースの削除
	RTRIM$	(〃)	右側スペースの削除
	SPACE$	(〃)	スペースの生成
	STRING$	(〃)	文字列の生成

■漢字の扱い

文字列中の漢字は，シフト JIS コードでメモリ上に上位バイト，下位バイトの順に格納されています．たとえば，次のようにです．

A$="日本語"

ところで，ASC,CHR$で扱う漢字のコードはシフト JIS コードではなく，漢字連続コードという特殊なもので，次のように対応しています．

漢字連続コード	シフト JIS コード	漢字
256	&H8140	スペース
257	&H8141	、
⋮	⋮	⋮
1666	&H889F	亜
⋮	⋮	⋮

5-9 数値処理

数値処理を行うためのステートメントと関数です.

	ステートメント/関数		機能
算術関数	ABS	(関数)	絶対値
	ATN	(〃)	アークタンジェント
	COS	(〃)	コサイン
	EXP	(〃)	指数
	FIX	(〃)	小数部切り捨て
	INT	(〃)	小数部切り捨て
	LOG	(〃)	自然対数
	SGN	(〃)	正・負符号
	SIN	(〃)	サイン
	SQR	(〃)	ルート
	TAN	(〃)	タンジェント
乱数	RANDOMIZE		乱数列の初期化
	RND	(関数)	乱数の発生

5-10 データ型の変換

数値データ間の型変換，数値データ↔文字データ間の型変換，マイクロソフト・バイナリ形式↔IEEE形式の変換を行うための関数です．

	関　数	機　能
数値	CINT	式の値を整数型に変換
	CLNG	式の値を倍長整数型に変換
	CSNG	式の値を単精度実数型に変換
	CDBL	式の値を倍精度実数型に変換
数値↕文字変換	CVD	8バイト文字列を倍精度実数値に変換
	CVI	2バイト文字列を整数値に変換
	CVL	4バイト文字列を倍長整数値に変換
	CVS	4バイト文字列を単精度実数値に変換
	HEX$	数値を16進文字列に変換
	MKD$	倍精度実数を8バイト文字列に変換
	MKI$	整数を2バイト文字列に変換
	MKL$	倍長整数を4バイト文字列に変換
	MKS$	単精度実数を4バイト文字列に変換
	OCT$	数値を8進文字列に変換
	STR$	数値を10進文字列に変換
	VAL	文字列を数値に変換
MBF↕IEEE	CVSMBF	4バイトのMBF文字列をIEEE形式の数値に変換
	CVDMBF	8バイトのMBF文字列をIEEE形式の数値に変換
	MKSMBF	IEEE形式の数値を4バイトのMBF文字列に変換
	MKDMBF	IEEE形式の数値を8バイトのMBF文字列に変換

■マイクロソフト・バイナリ形式とIEEE 形式

浮動小数点数の内部データ形式は，マイクロソフト・バイナリ形式とIEEE 形式とがあります．従来のマイクロソフトBASICはマイクロソフト・バイナリ形式を用いていましたが，Quick BASICではIEEE 形式を採用しました．IEEE 形式は，インテルの数値演算コ・プロセッサ8087との相性のよい標準的なデータ形式です．

以下に単精度実数のマイクロソフト・バイナリ形式と，IEEE 形式のデータ表現を示します．

●マイクロソフト・バイナリ形式

A	B	C	D
s ¦ f_1～f_7	f_8～f_{15}	f_{16}～f_{23}	e_8～e_1

s：符号，f_1～f_{23}：仮数部，e_8～e_1：指数部

→ メモリ格納イメージ

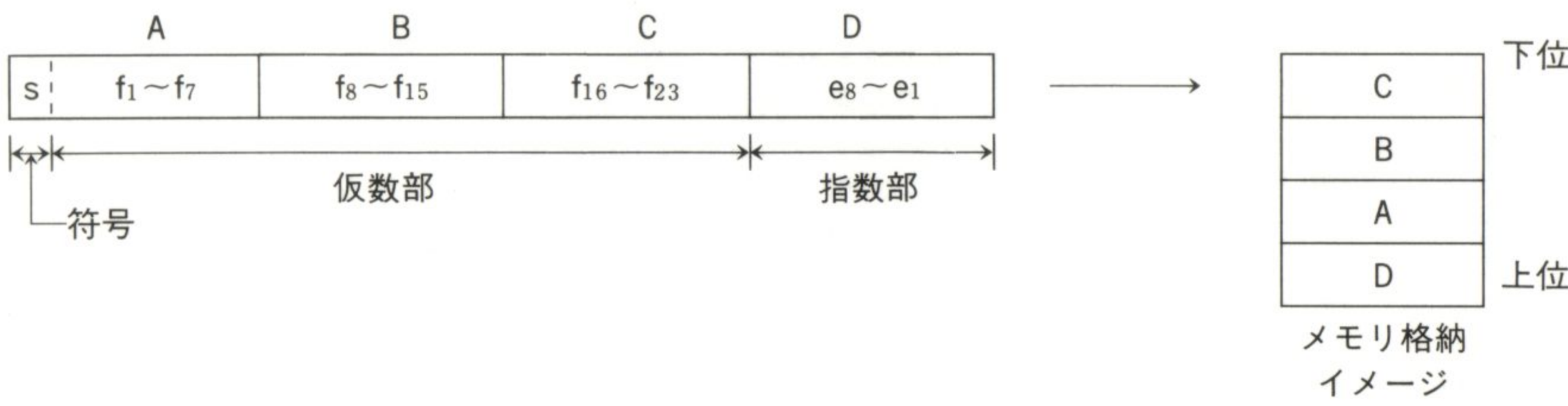
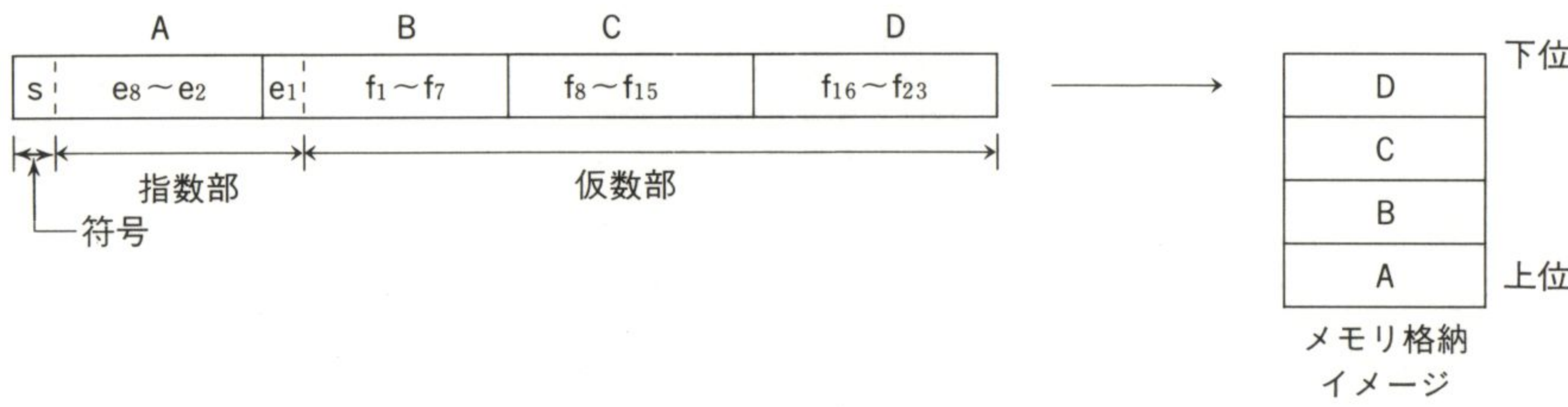
C	下位
B	
A	
D	上位

●IEEE 形式

A		B	C	D
s ¦ e_8～e_2	e_1 ¦	f_1～f_7	f_8～f_{15}	f_{16}～f_{23}

s：符号，e_8～e_1：指数部，f_1～f_{23}：仮数部

→ メモリ格納イメージ

D	下位
C	
B	
A	上位

IEEE 形式の数値を，マイクロソフト・バイナリ形式のメモリ格納イメージの文字列に変換したり，その逆を行う関数がCVSMBF,CVDMBF,MKSMBF，MKDM-BFです．

5−11 エラー処理とトラッピング

エラーが発生したときのエラー情報の取得，エラーが発生したときに分岐するトラッピングルーチンの設定，イベントが発生したときに分岐するトラッピングルーチンの設定を行うステートメントと関数です．

	ステートメント/関数		機　　能
エラー情報	ERL	(関数)	エラーがあった行番号の取得
	ERR	(〃)	エラーコードの取得
	ERDEV	(〃)	デバイスのエラーコードの取得
	ERDEV$	(〃)	エラーが発生したデバイス名の取得
	ERROR		エラーコードに対応したエラーメッセージの表示
エラートラッピング	ON ERROR GOTO		エラー・トラッピングルーチンの設定
	RESUME		エラー・トラッピングルーチンからの復帰
イベントトラッピング	ON event GOSUB		イベント・トラッピングルーチンの設定
	event ON/OFF/STOP		イベント・トラッピングの起動/無効/中断
	RETURN		イベント・トラッピングルーチンからの復帰

■エラー・トラッピング

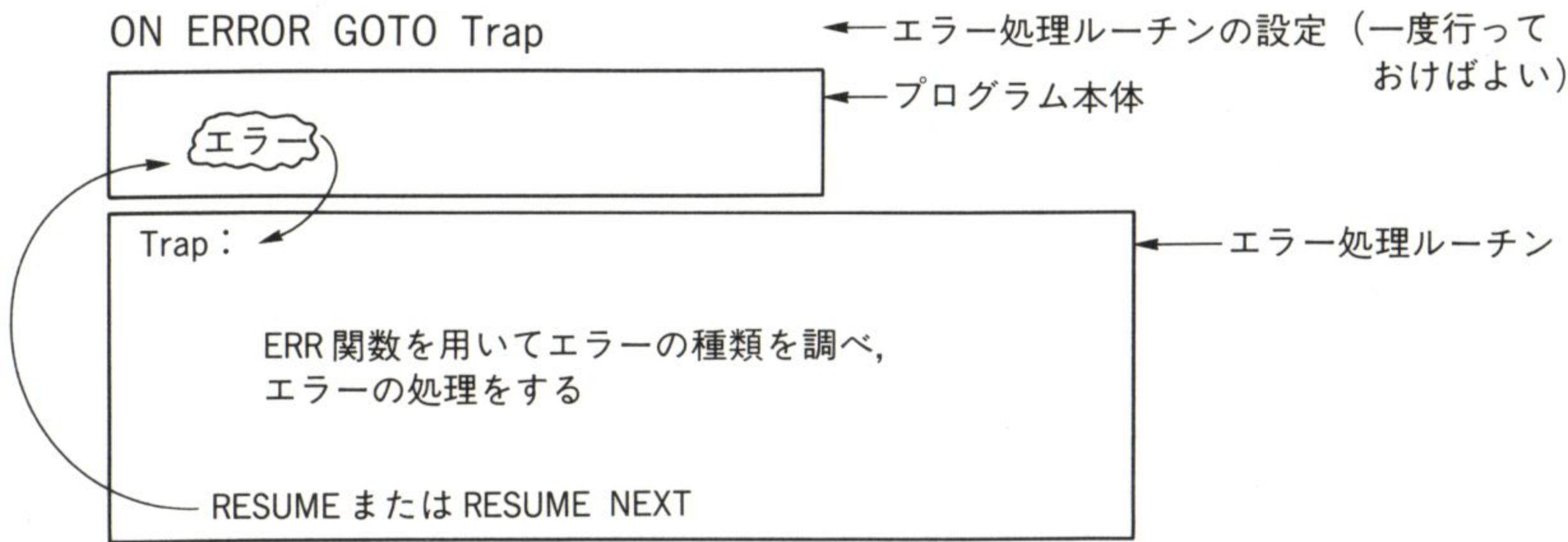
プログラム実行中にエラーが発生した場合，プログラムを中断して，割り込みによりエラー処理ルーチンに分岐させることをエラー・トラッピングといいます．エラー処理ルーチンの設定は，ON ERROR GOTO 文を用いて一度行っておけば，エラーが発生するたびに，設定した処理ルーチンへ分岐します．

ON ERROR GOTO Trap　←エラー処理ルーチンの設定（一度行っておけばよい）

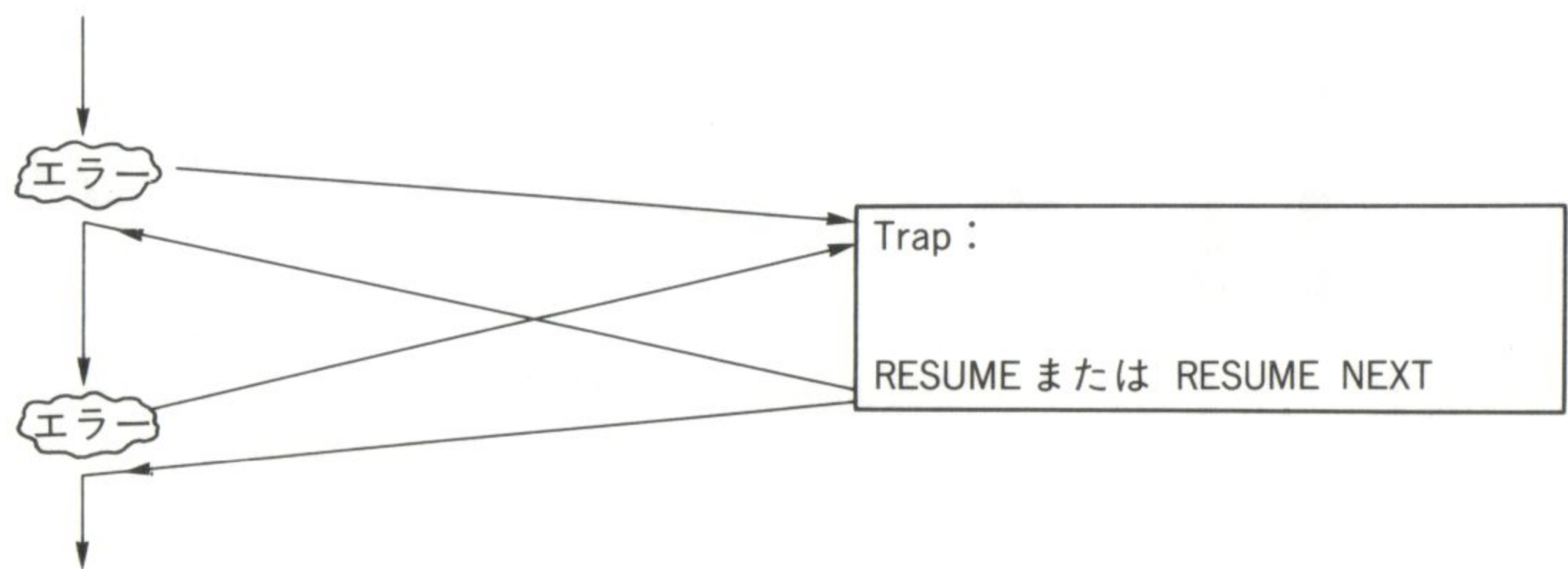
エラー　←プログラム本体

Trap：　←エラー処理ルーチン

ERR 関数を用いてエラーの種類を調べ，
エラーの処理をする

RESUME または RESUME NEXT

RESUME はエラーのあった文に戻るのに対し，RESUME NEXT はエラーのあった文の次の文に戻ります．

ON ERROR GOTO Trap（一度行っておけばよい）

エラー
Trap：
RESUME または RESUME NEXT
エラー

■イベント・トラッピング

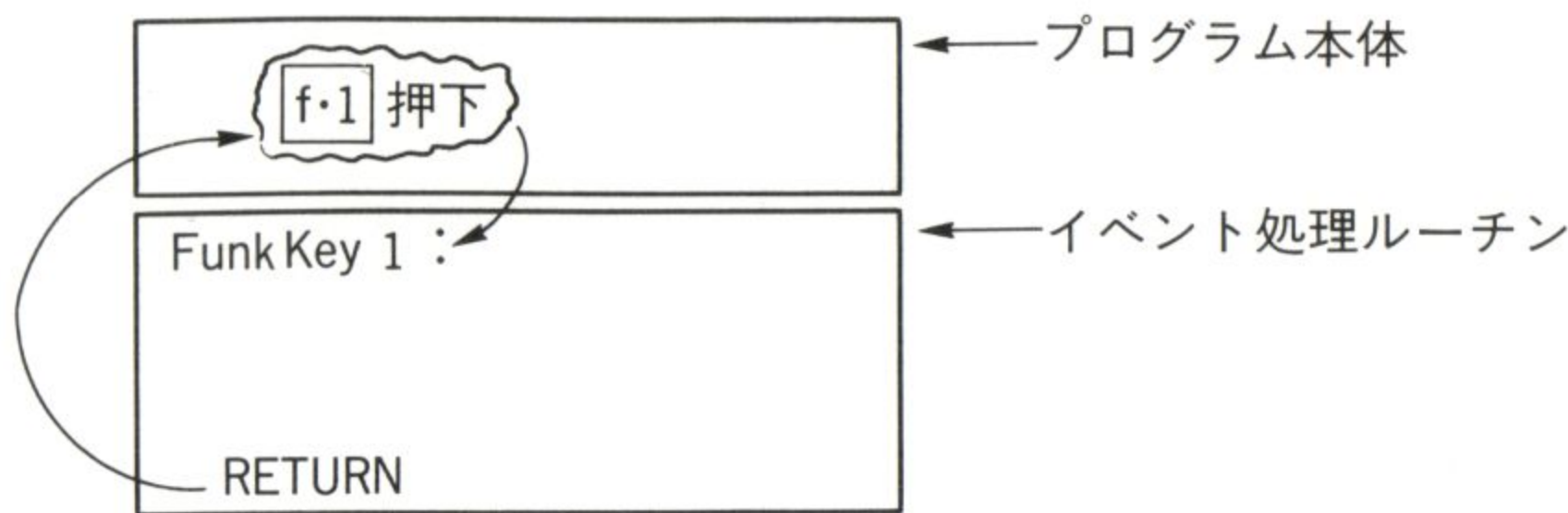
エラー・トラッピングと同様な処理を，イベントの発生により行わせることをイベント・トラッピングといいます．イベント・トラッピッグ処理ルーチンは，ON event GOSUB 文により設定することができます．event としては，次の 3 つが指定できます．

COM(portnumber) … シリアルポートにデータを受信したとき
KEY(keynumber) … 指定されたキーが押されたとき
TIMER(interval) … 指定秒が経過したとき

なお，イベント・トラッピングを可能にするためには，event ON によりトラッピングを有効にしておきます．

たとえば，ファンクションキーの f･1 が押された場合の処理ルーチンは次のようになります．

```
ON KEY(1) GOSUB  FunkKey1
KEY(1) ON
```

f･1 押下
←プログラム本体
FunkKey 1：
←イベント処理ルーチン
RETURN

■複数モジュールでのトラッピング

複数モジュールでのトラッピング処理は，エラー・トラッピングとイベント・トラッピングで多少異なります．

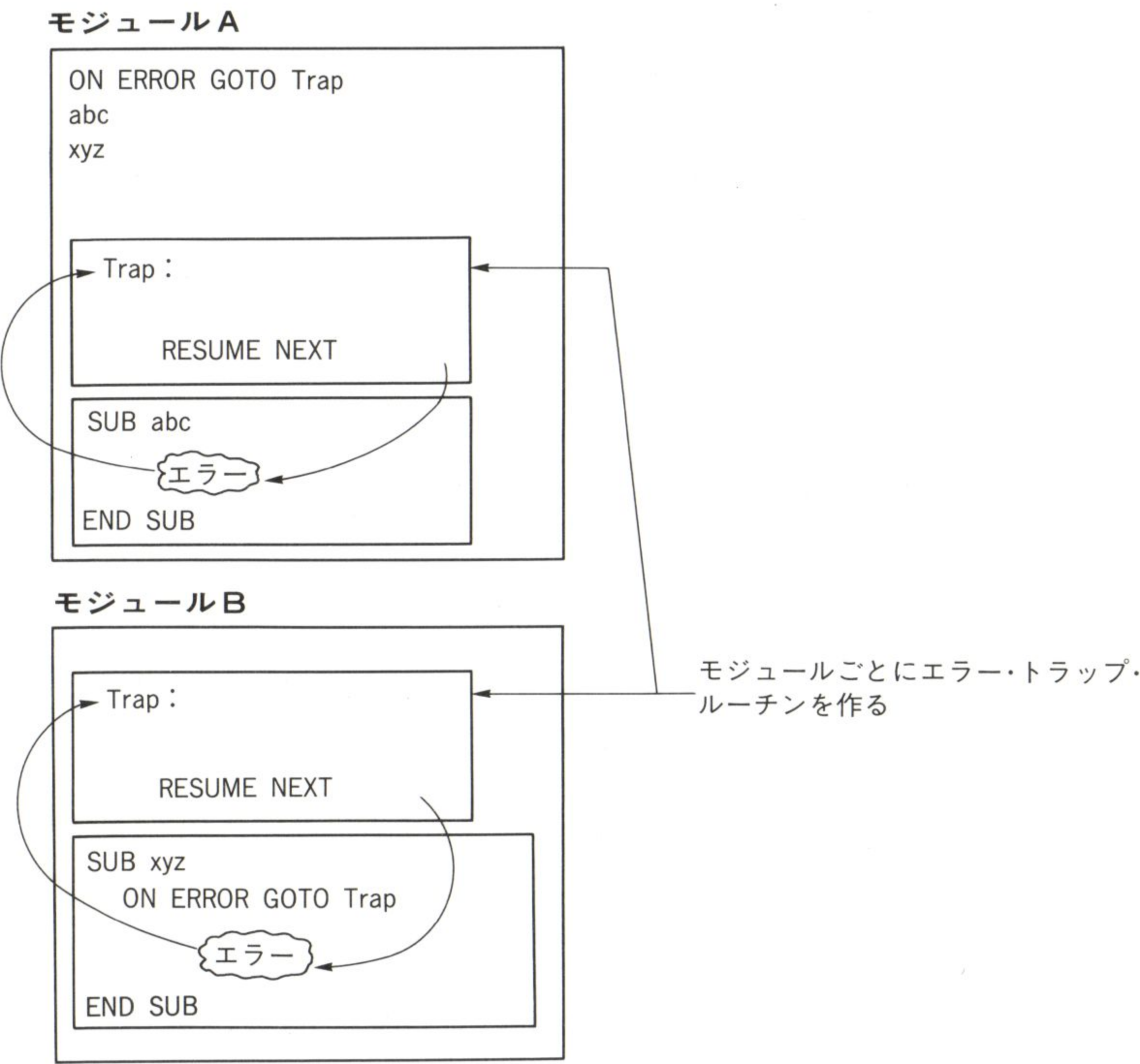
エラー・トラッピングでは，ほかのモジュールのトラッピングルーチンを呼び出した場合，戻ってこれませんので，モジュール単位で固有のトラッピングルーチンを設定しなければなりません．

モジュールA

```
ON ERROR GOTO Trap
abc
xyz

Trap：

    RESUME NEXT

SUB abc
    エラー
END SUB
```

モジュールB

```
Trap：

    RESUME NEXT

SUB xyz
  ON ERROR GOTO Trap
    エラー
END SUB
```

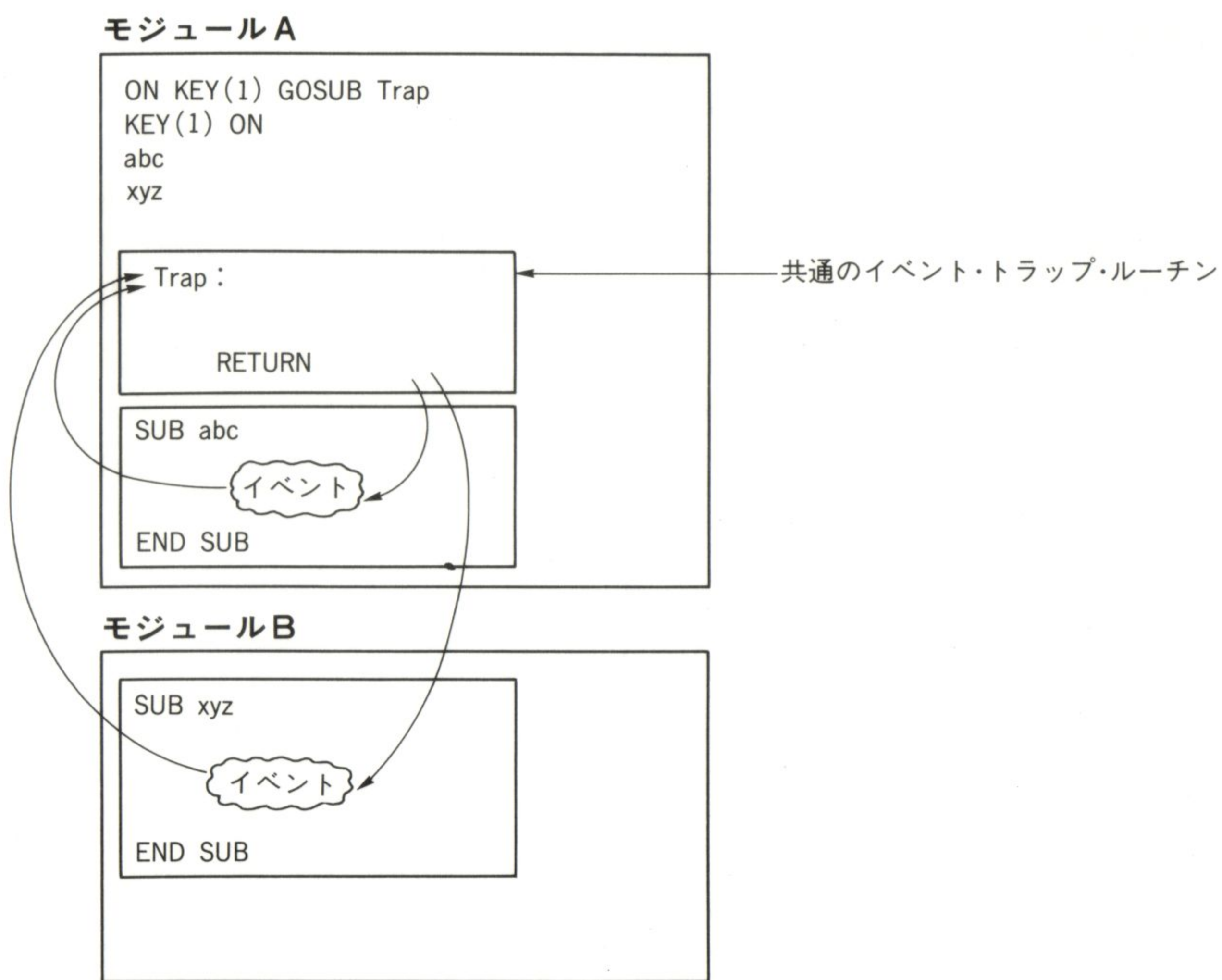
モジュールごとにエラー・トラップ・ルーチンを作る

これに対し，イベント・トラッピングでは，ほかのモジュールのトラッピングルーチンを使用することができ，トラッピングルーチンはモジュール間で共通に利用されます．

モジュールA

```
ON KEY(1) GOSUB Trap
KEY(1) ON
abc
xyz

  Trap:

      RETURN

  SUB abc
      イベント
  END SUB
```

共通のイベント・トラップ・ルーチン

モジュールB

```
  SUB xyz
      イベント
  END SUB
```

■コンパイル・オプション

エラー・トラッピングを行うプログラムを，BC でコンパイルするときには次のオプションを指定します．

オプション	機　　　能
/E	ON ERROR GOTO，RESUME line 命令がプログラム中にある場合に指定
/V /W	ON event GOSUB，event ON 命令がプログラム中にある場合に指定. /V はステートメントごとにトラッピングを行い, /W は行ごとにトラッピングを行う
/X	RESUME，RESUME NEXT，RESUME 0命令がプログラム中にある場合に指定

なお，QB 上から BC を起動する場合は，自動的にこれらのオプションが選択されます．

5−12 時間

時間の設定/取得を行ったり，タイマ・トラッピングを行うためのステートメントと関数です．

	ステートメント/関数	機　　能
時間の設定/取得	DATE$　（関数）	現在の日付の取得
	DATE$	現在の日付の設定
	TIME$　（関数）	現在の時刻の取得
	TIME$	現在の時刻の設定
	TIMER　（関数）	午前0時からの経過秒の取得
タイマトラッピング	ON TIMER GOSUB	時間経過によりサブルーチンに分岐
	TIMER ON/OFF/STOP	タイマ・イベント・トラッピングの開始/禁止/停止

5-13 メモリ操作

メモリのリード/ライト，メモリの取得/解放などのメモリ操作を行うためのステートメントと関数です．

ステートメント/関数	機　　能
DEF SEG	セグメント・アドレスの設定
FRE　（関数）	使用可能なメモリサイズの取得
PEEK　（ 〃 ）	メモリから1バイトをリード
POKE	メモリに1バイトをライト
SETMEM	far ヒープで使用するメモリ量の設定

■セグメント値とオフセット値

8086では，1MBのメモリ空間を64KBのセグメントに分割して管理しています．セグメントの先頭アドレスをセグメント値といい，物理アドレスの下位4ビットを削除したものを用います．セグメント内のアドレスは，セグメント先頭からの距離で表し，これをオフセット値といいます．

オフセット値は，0000H～FFFFHまでの値をとるので，1つの論理セグメントが扱うことのできる最大メモリは，64KBとなります．

メモリ・アクセスするには，まず，DEF SEGによりセグメント値を指定します．その後，PEEK，POKE，BLOAD，BSAVE，CALL ABSOLUTEで指定するアドレスは，DEF SEGで設定したセグメント先頭からのオフセット値です．

DEF SEGでセグメント値を指定しなければ，BASICデータセグメント(レジスタDSで示される)が使用されます．

5-14 DOS環境

MS-DOSのコマンドラインの取得，環境変数の設定/取得，システムコールなどを行うステートメントと関数です．

ステートメント/関数	機能
CALL INT86OLD	DOSのシステムコール
CALL INT86XOLD	DOSのシステムコール
CALL INTERRUPT	DOSのシステムコール
CALL INTERRUPTX	DOSのシステムコール
COMMAND$ （関数）	コマンドライン引数文字列の取得
ENVIRON$ （ 〃 ）	DOS環境文字列の取得
ENVIRON	DOS環境文字列の設定
SHELL	実行中のプログラムを一時中断して他のプログラムを実行

■ソフトウェア割り込みとシステムコール

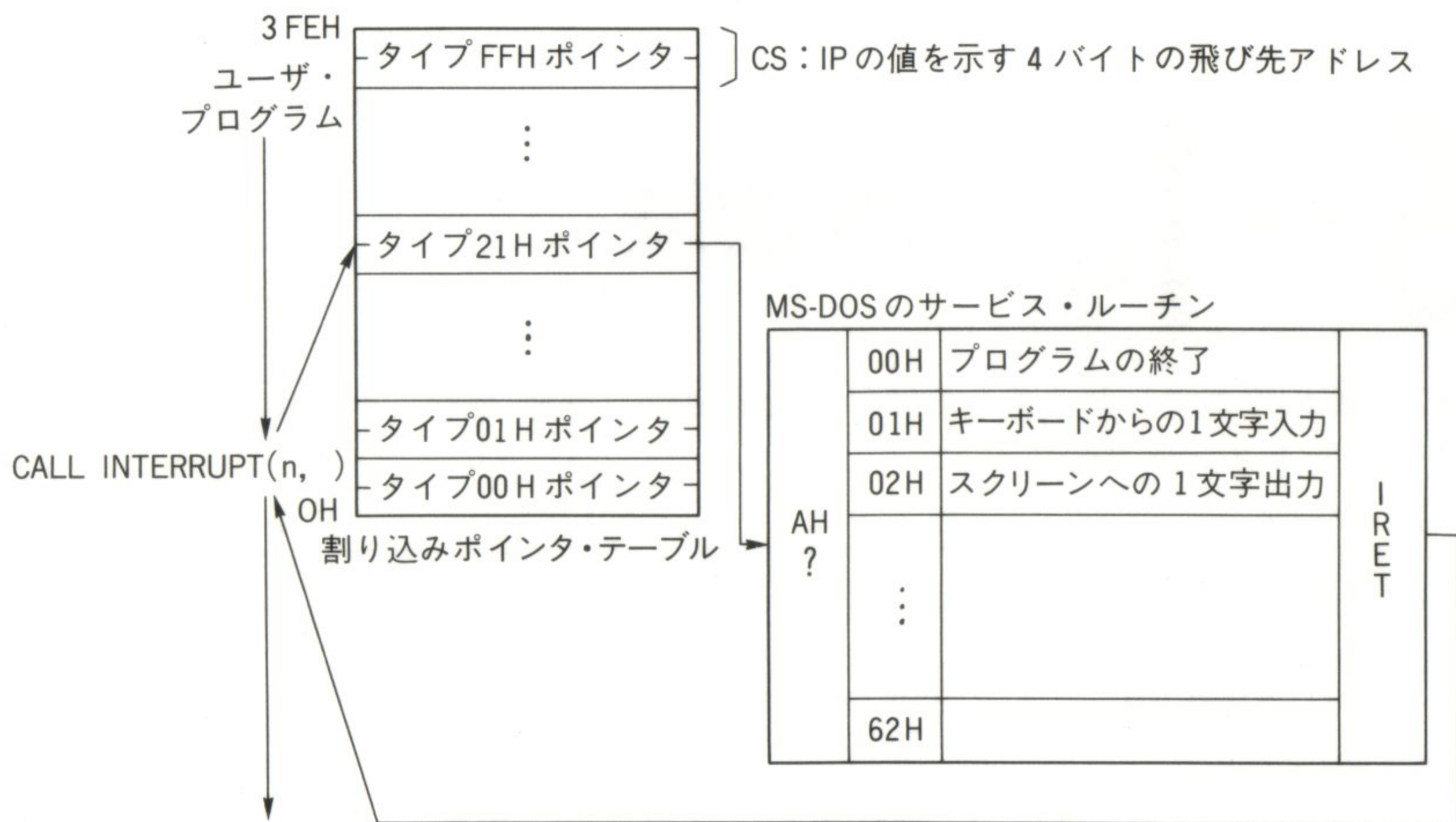
現在実行中の処理を中断して，別のサービス・ルーチンを実行したい場合に，ソフトウェア割り込みという手法を用います．

Quick BASICからソフトウェア割り込みを行うには，

```
CALL INTERRUPT(n, )
```

を行います．nには0～255を指定できます．CALL INTERRUPT(n,)が発生すると，メモリ0番地から設定されている割り込みポインタ・テーブルの第nポインタが参照され，それによって示される割り込みサービス・ルーチンへ分岐し，割り込み処理終了後，分岐してきたもとの位置に復帰します．

ソフトウェア割り込みにより，MS-DOSシステムをコールするものをシステムコールといいます．システムコールの21H番は，約90種類のファンクション(サブルーチン)を持っています．

3FEH
ユーザ・
プログラム
タイプFFHポインタ
CS：IPの値を示す4バイトの飛び先アドレス
タイプ21Hポインタ
MS-DOSのサービス・ルーチン
00H
プログラムの終了
01H
キーボードからの1文字入力
02H
スクリーンへの1文字出力
62H
AH
?
IRET
タイプ01Hポインタ
タイプ00Hポインタ
CALL INTERRUPT(n,)
0H
割り込みポインタ・テーブル

5−15 グラフィック

グラフィック処理を行うためのステートメントと関数です.

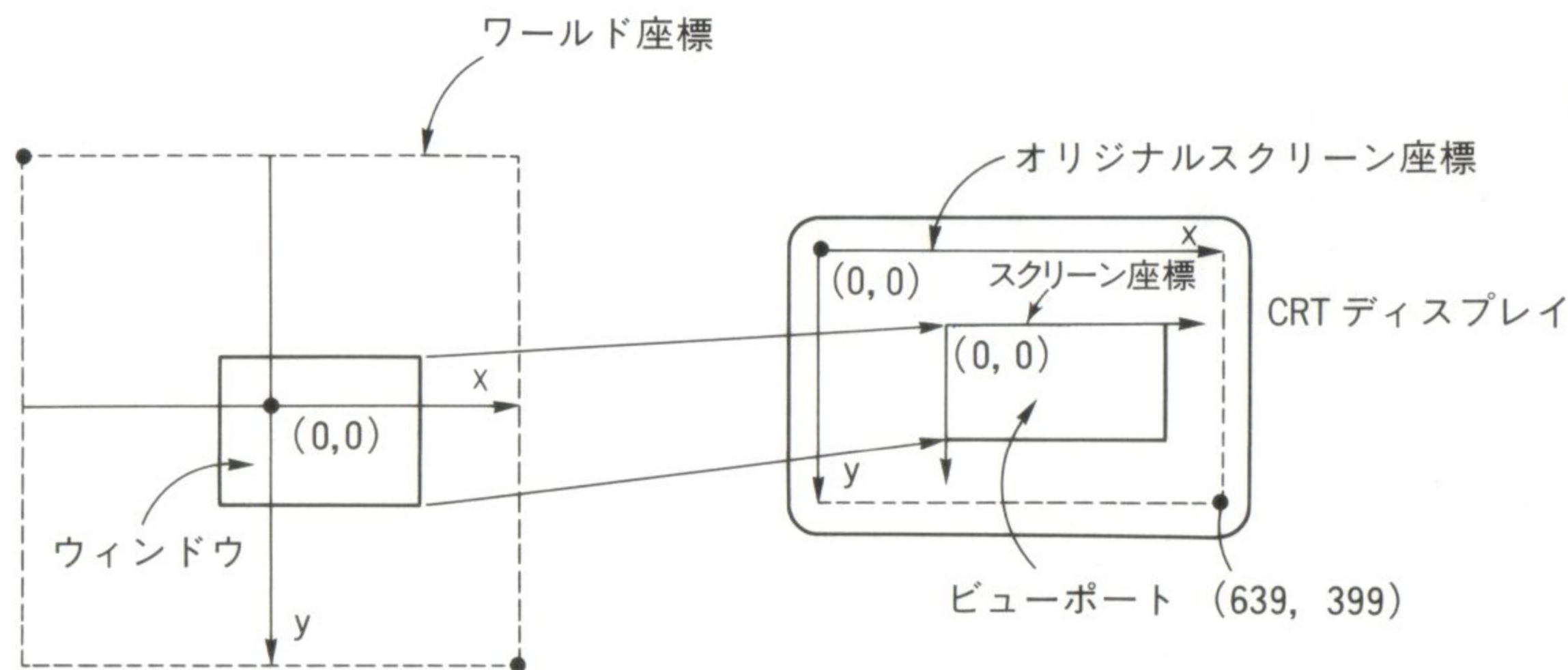
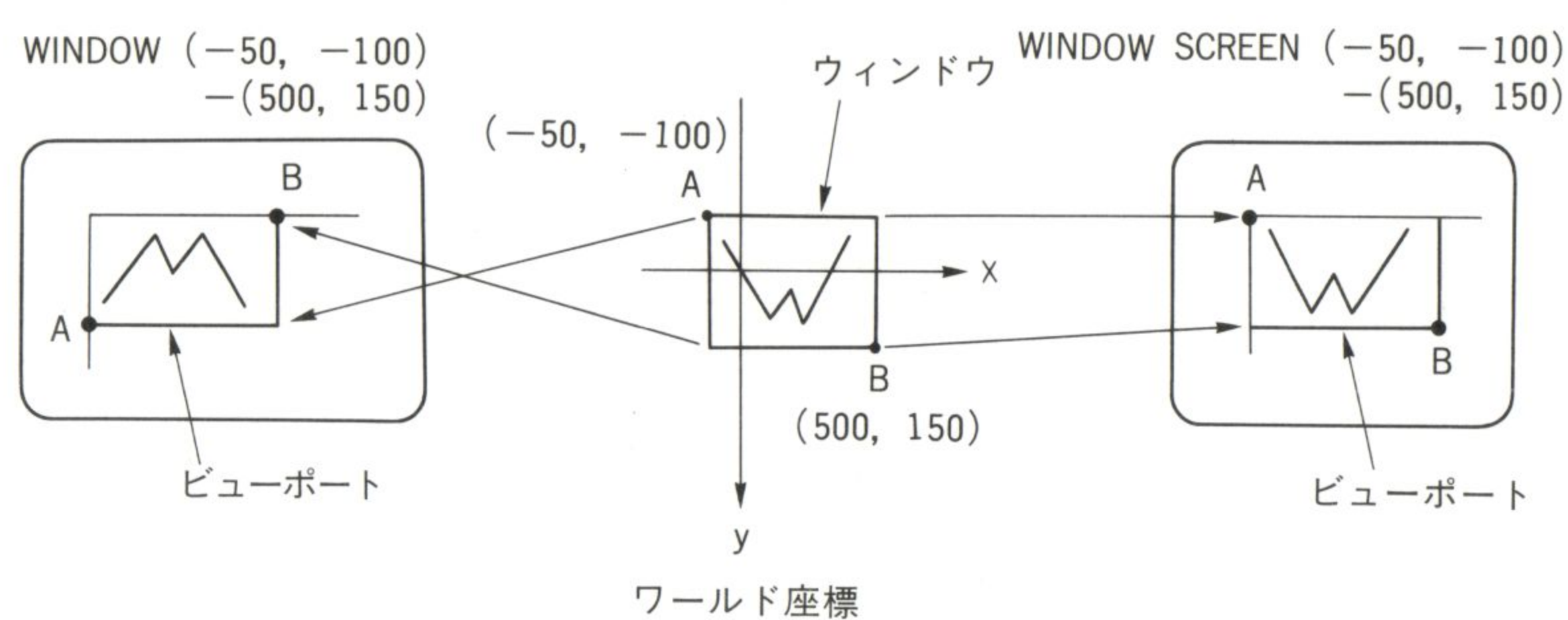
	ステートメント/関数		機　能
画面設定	CLS		画面のクリア
	SCREEN		画面モードの設定
	WINDOW		ウィンドウの設定
	VIEW		ビューポートの設定
画面情報	PMAP	(関数)	論理座標↔物理座標の変換
	POINT	(〃)	描画現在位置の取得
	POINT	(〃)	指定位置の色の取得
表示色	COLOR		表示色の指定
	PALETTE		パレットの設定
	PALETTE USING		パレットの設定
描画	CIRCLE		円の描画
	DRAW		マクロコマンドによる描画
	GET		画面イメージの保存
	LINE		直線の描画
	PAINT		閉ループ内のぬりつぶし
	PCOPY		スクリーンページを別のページにコピー
	PRESET		指定点のドットの消去
	PSET		指定点にドットの表示
	PUT		保存イメージの描画

■グラフィック座標

グラフィック処理では，ワールド座標，オリジナルスクリーン座標，スクリーン座標という3つの座標を扱っています.

LINEやCIRCLEでの作図はワールド座標に行われます．ワールド座標のどの範囲をCRTディスプレイへの表示範囲にするかをWINDOW文で指定します．この範囲をウィンドウと呼びます.

CRTディスプレイ上の表示範囲はVIEW文で指定します．この範囲をビューポートと呼びます．CRTディスプレイの左隅上の点を(0，0)とする座標をオリジナルスクリーン座標と呼びます.

CLS または CLS 1 で消される範囲はビューポートの中だけです．

Quick BASIC では，WINDOW 文によるウィンドウの切り方に 2 通りの方法があります．WINDOW 文に SCREEN オプションを付けると，下方を Y 座標の正の向きとするワールド座標をそのままの向きでビューポートに取り出し，SCREEN オプションを付けないと，Y 方向を逆転してビューポートに取り出します．

■画面モード

画面モードはテキスト・モード，グラフィック・モード，テキスト・モードとグラフィック・モードを同時に表示するスーパーインポーズ・モードに分かれます．さらに，グラフィック・モードはドット構成で，640×200(低解像)と640×400(高解像)に分かれます．

これらの画面モードは SCREEN 文で指定します．

```
SCREEN  画面モード
         ├81        … テキスト
         ├84        … 640×200  スーパーインポーズ
         ├87        … 640×400  グラフィック
         └88または0 … 640×400  スーパーインポーズ
```

SCREEN 0が実行された直後のウィンドウとビューポートは,

```
WINDOW SCREEN  (0, 0)-(639, 399)
VIEW  (0, 0)-(639, 399)
```

が指定されたのと同じ状態です.

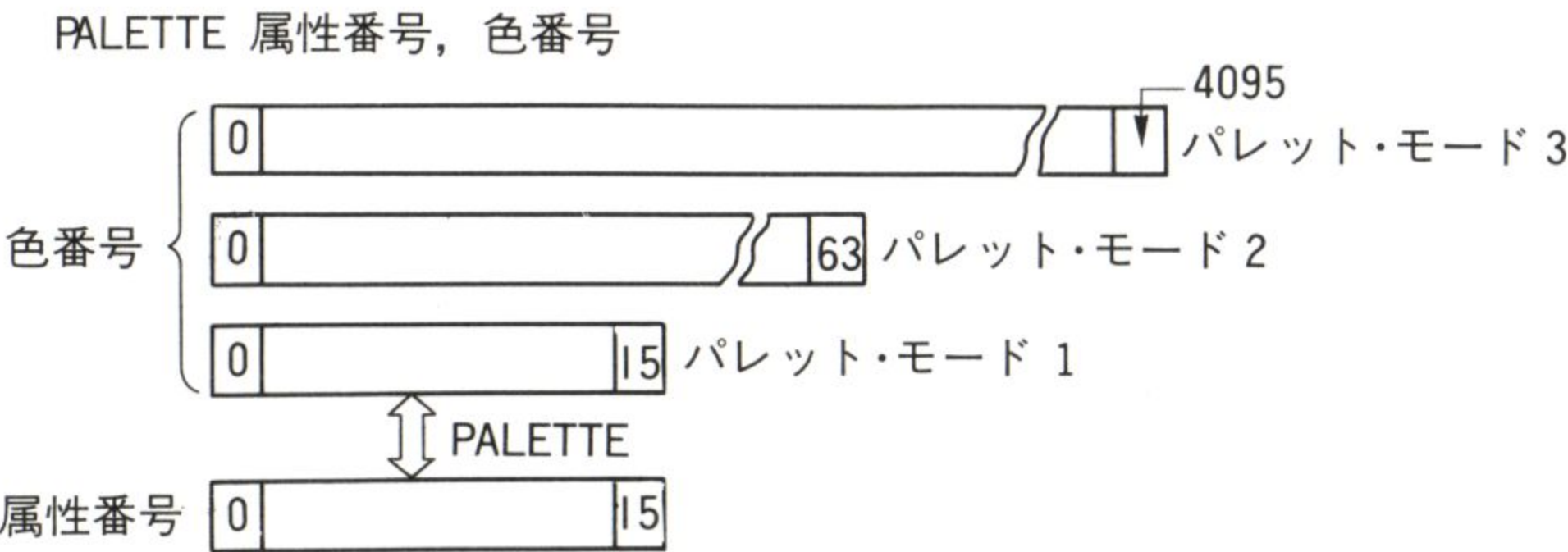
■表示色

使用できる表示色の種類はSCREEN文で指定するパレット・モードにより異なります.

```
SCREEN  画面モード, パレット・モード
                      ├1…   16色中16色. 色番号0～  15
                      ├2…   64色中16色. 色番号0～  63
                      └3…4096色中16色. 色番号0～4095
```

LINEやCIRCLEで指定する色は,実際の色番号ではなく,属性番号(0～15)で行います.
属性番号と色番号はPALETTE文により対応付けられています.

PALETTE 属性番号, 色番号

色番号
0 … 4095 パレット・モード3
0 … 63 パレット・モード2
0 … 15 パレット・モード1
⇕ PALETTE
属性番号 0 … 15

パレット・モード1(16色モード)のときの,色番号と色の対応は以下のとおりです.

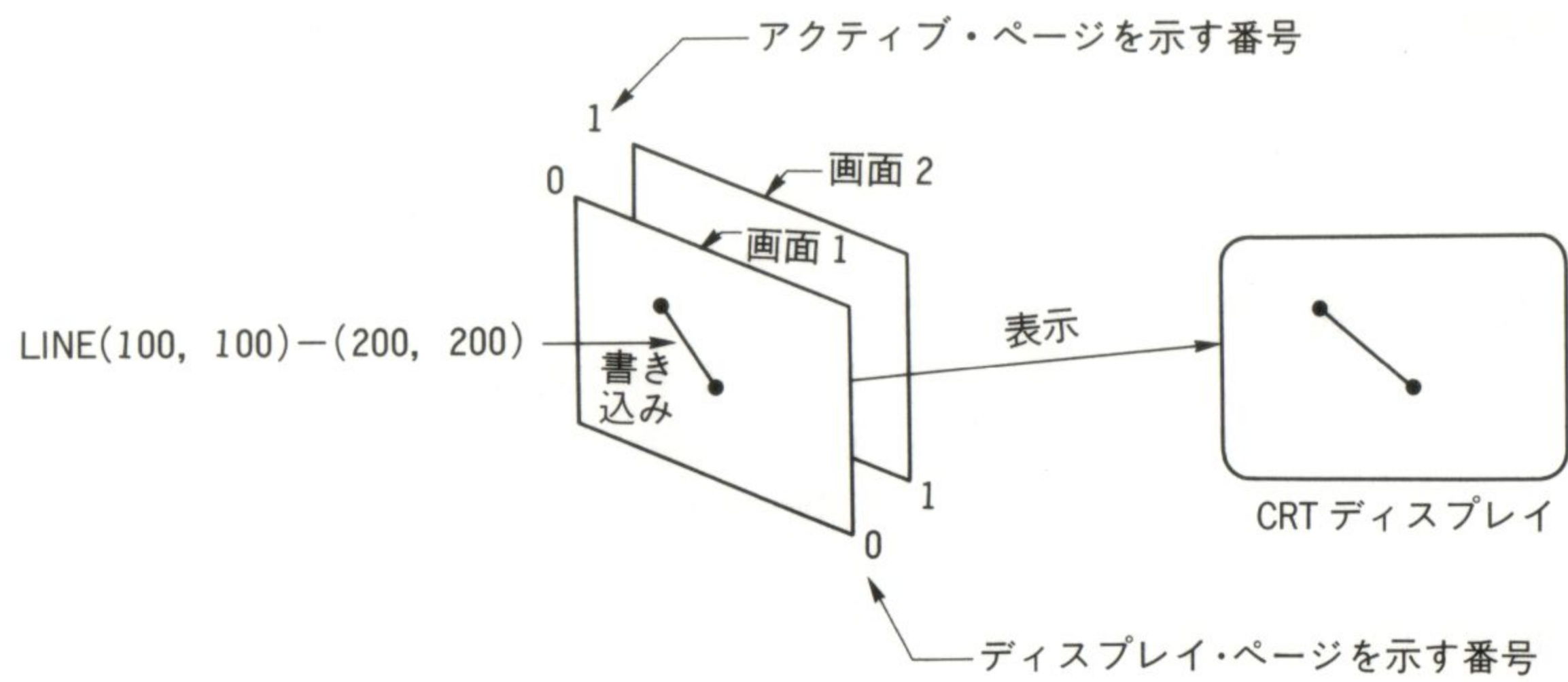
色番号	0	1	2	3	4	5	6	7	
色	黒	青	緑	水	赤	紫	黄	白	←明るい色

色番号	8	9	10	11	12	13	14	15	
色	黒	青	緑	水	赤	紫	黄	白	←暗い色

注)デジタル・ディスプレイでは,0～7だけが表示できます.

■アクティブ・ページとディスプレイ・ページ

アクティブ・ページはグラフィックデータを書き込む画面，ディスプレイ・ページはCRTディスプレイに表示する画面で，SCREEN文により指定します．

SCREEN　画面モード，パレット・モード，アクティブ・ページ，ディスプレイ・ページ

注）640×200モードでは 0 ～ 3 の 4 画面が指定できます．

5-16 その他

いままでの分類に入らないステートメントです.

	ステートメント	機　　能
一般	BEEP LET READ～DATA REM RESTORE SWAP	ビープ音の発生 代入 プログラム中のデータを読む コメント リードデータ位置の設定 変数の内容を交換
機械語	BLOAD BSAVE CALL ABSOLUTE	メモリイメージのファイルをメモリにロード メモリイメージをファイルにセーブ 機械語プロシージャのコール
実行	CHAIN RUN RESET SYSTEM TRON TROFF	プログラムのチェイン プログラムの実行 すべてのディスク・ファイルをクローズ プログラムを終了し，OS に戻る トレース・オン トレース・オフ

第6章

ステートメント・関数リファレンス

ABS （絶対値） 関数

書式 ABS(x)

機能 引数 x の絶対値を返します．

ASC （文字コードの取得） 関数

書式 ASC(str)

　　　　↑文字列

機能 文字列 str の先頭 1 文字のコードを返します．その文字が ASCII 文字なら ASCII コード，漢字なら 2 バイトの漢字コードです．

例

```
PRINT "A  -->"; ASC("A")
PRINT "亜 -->"; ASC("亜")
```

```
A  --> 65
亜 --> 1666
```

ATN （アークタンジェント） 関数

書式 ATN(x)

機能 x のアークタンジェントを返します．結果は$-\pi/2$〜$\pi/2$の範囲です．

BEEP （ビープ音）

書式 BEEP

機能 ビープ音を発生させます．

BLOAD（メモリイメージデータのメモリへのロード）

書式

```
BLOAD    filename[,offset]
```

- offset：ロードするメモリの先頭オフセット値
- filename：ファイル名

機能

BSAVEでセーブしたメモリイメージのファイルを，offsetで示されるメモリにロードします．offsetを省略すると，BSAVEでセーブするときに指定したアドレスが使われます．

BSAVE（メモリイメージデータのファイルへのセーブ）

書式

```
BSAVE    filename,offset,n
```

- n：セーブするメモリのサイズ(バイト)
- offset：セーブするメモリの先頭オフセット値
- filename：ファイル名

機能

offset以後のnバイトのメモリのデータを，filenameで示されるファイルにセーブします．nは0～65535までです．

例

PC-9801シリーズのグラフィックデータは，A8000番地以後のVRAMという領域に格納されています．

このデータをBSAVEでファイルにセーブし，BLOADで再びVRAM上にロードする方法を示します．

```
' ---/* グラフイック画面のセーブ */---
DEF SEG = &HA800
BSAVE "b:gdata1.dat", &H0, &HFFFF
DEF SEG = &HB800
BSAVE "b:gdata2.dat", &H0, &H8000
DEF SEG = &HE000
BSAVE "b:gdata3.dat", &H0, &H8000

' ---/* グラフイック・データのロード */---
DEF SEG = &HA800
BLOAD "b:gdata1.dat", &H0
DEF SEG = &HB800
BLOAD "b:gdata2.dat", &H0
DEF SEG = &HE000
BLOAD "b:gdata3.dat", &H0
```

アドレス	内容
A8000	プレーン1 (Blue)
B0000	プレーン2 (Red)
B8000	プレーン3 (Green)
E0000	プレーン4 (輝度)

CALL　　　　　　（プロシージャのコール）

書式

```
CALL   name[(arg,…)]
       ↑         ↑
       │         └引数
       └プロシージャ名
```

機能

nameで示されるプロシージャをコールします．DECLARE文によりプロシージャnameが宣言されていれば，CALLを付けずにプロシージャ名を書くだけで，プロシージャをコールすることができます．

CALL/CALLS　（他言語プロシージャのコール）

書式

```
CALL    name [ [ {BYVAL | SEG} ] arg,…]
        ↑        ↑                ↑
        │        │                └引数
        └プロシージャ名 └引数渡しの方法

CALLS  name[(arg,…)]
       ↑         ↑
       │         └引数
       └プロシージャ名
```

機能

他言語プロシージャをコールします．引数の前にBYBALまたはSEGを指定して，引数渡しの方法を指定することができます．指定しなければ，near参照により引数が渡されます．

BYVAL … **値による渡し**
SEG … **far参照**

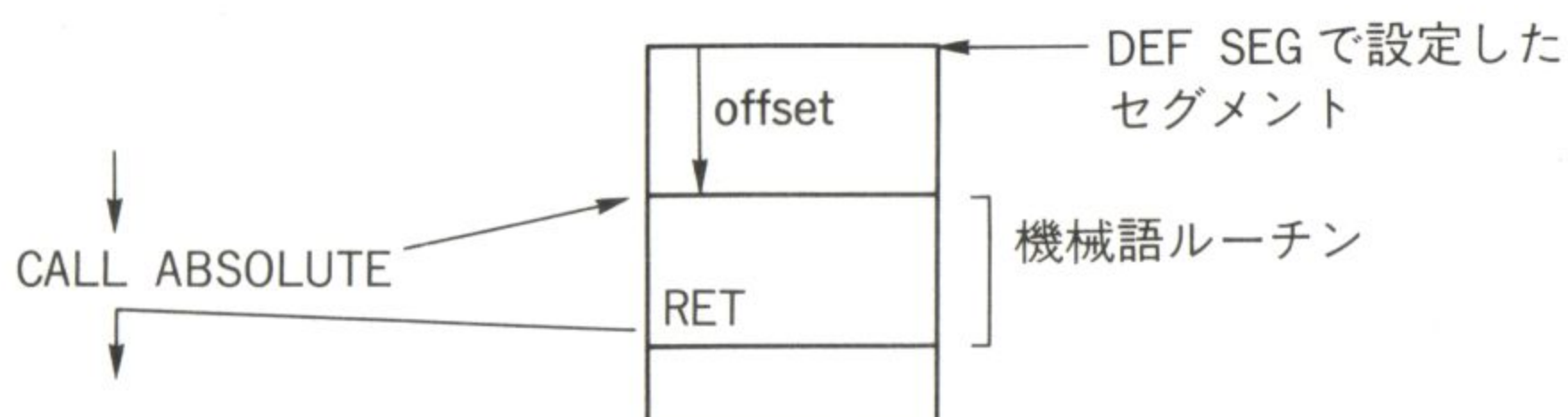
CALLSでは，引数はfar参照で渡されます．

CALL ABSOLUTE（機械語プロシージャのコール）

書式

```
CALL   ABSOLUTE([ arg,…,] offset)
                  ↑        ↑
                  │        └プロシージャが置かれている
                  └引数      アドレス(オフセット値)
```

機能

offsetで示されるメモリに置かれている，機械語ルーチンをコールします．セグメント値は，DEF SEGを用いて設定しておいてください．

CALL INT86OLD/CALL INT86XOLD (DOSシステムコール)

書式　CALL INT86OLD/INT86XOLD(n, inarray(), outarray())

- n ― 割り込み番号
- inarray() ― レジスタに渡すデータを入れた配列
- outarray() ― システムコール後に返されるレジスタの値を入れる配列

機能

割り込み番号nのソフトウェア割り込みを行います．inarray( )とoutarray( )は，レジスタとの間のデータ授受を行うための整数配列で，次のようにレジスタと対応しています．

	レジスタ	
inarray(0)	AX	
(1)	BX	
(2)	CX	
(3)	DX	
(4)	BP	
(5)	SI	
(6)	DI	
(7)	FLAGS	
(8)	DS	INT86XOLDのときだけ使用する
(9)	ES	

INT86OLDとINT86XOLDの違いは，レジスタDS,ESのデータ授受が行えるか否かということです．

INT86OLDとINT86XOLDプロシージャは，クイックライブラリ(QB.QLB)，およびライブラリ(QB.LIB)に含まれていますので，QB環境で使用するにはQB起動時に，

```
QB  /L
```

として，クイックライブラリ(QB.QLB)を組み込んでおかなければなりません．

例

```
' ---/* システム時間の取得 */---

DIM inarray%(9), outarray%(9)

inarray%(0) = &H2C00:    時間の取得                      ┐← システムコール
CALL INT86OLD(&H21, inarray%(), outarray%())              ┘

IF (inarray%(7) AND &H1) = 0 THEN ←──── システムコールにエラーが発生すると
                                          FLAGSの最下位ビットがセットされる
        h% = outarray%(2) ¥ &H100:  ' 時 ←─ CXの上位バイトを取り出す
        m% = outarray%(2) AND &HFF: ' 分 ←─ CXの下位バイトを取り出す
        s% = outarray%(3) ¥ &H100:  ' 秒
        PRINT "Time="; h%; ":"; m%; ":"; s%
ELSE
        PRINT "Time error"
END IF
```

CALL INTERRUPT/CALL INTERRUPTX (DOSシステムコール)

書式

```
CALL INTERRUPT/INTERRUPTX(n, inregs, outregs)
```

- n：割り込み番号
- inregs：レジスタに渡すデータを入れたレコード型変数
- outregs：システムコール後に返されるレジスタの値を入れるレコード型変数

機能

割り込み番号nの，ソフトウェア割り込みを行います．inregsとoutregsは，レジスタとの間のデータ授受を行うためのレコード型変数で，その型RegTypeはQB.BIの中で型宣言されています．

```
TYPE   RegType
  AX    AS    INTEGER
  BX    AS    INTEGER
  CX    AS    INTEGER
  DX    AS    INTEGER
  BP    AS    INTEGER
  SI    AS    INTEGER
  DI    AS    INTEGER
  FLAGS    AS   INTEGER
  DS    AS    INTEGER   | INTERRUPTXのときだけ
  ES    AS    INTEGER   | 使用する
END TYPE
```

INTERRUPTとINTERRUPTXの違いは，レジスタDS,ESのデータ授受が行えるか否かということです．

INTERRUPTとINTERRUPTXプロシージャは，クイックライブラリ(QB.QLB)，およびライブラリ(QB.LIB)に含まれていますので，QB環境で使用するには,QB起動時に,

```
QB /L
```

として，クイックライブラリ(QB.QLB)を組み込んでおかなければなりません．

CALL INT86OLD/CALL INT86XOLDとCALL INTERRUPT/CALL INTERRUPTXは機能的には同じものですが，レジスタ引数に前者は配列，後者はレコード型を用いている点が異なります．データをスマートに記述するためには，後者を用いるのがよいでしょう．

例

システムコール(21H)のファンクション2CHを用いて，システム時間を取得するものです。

```
'$INCLUDE: 'qb.bi' ←――― RegType型がこの中で宣言されているので必ず取り込む

DIM InRegs AS RegType, OutRegs AS RegType

InRegs.ax = &H2C00:  ' 時間の取得              ┐
CALL INTERRUPT(&H21, InRegs, OutRegs)          ┘←― システムコール

IF (OutRegs.flags AND &H1) = 0 THEN  ←―――― システムコールにエラーが発生すると
                                              flagsの最下位ビットがセットされる
        h% = OutRegs.cx ¥ &H100:  ' 時 ←―┐
        m% = OutRegs.cx AND &HFF: ' 分 ←┐└― CXの上位バイトを取り出す
        s% = OutRegs.dx ¥ &H100:  ' 秒  └―― CXの下位バイトを取り出す
        PRINT "Time="; h%; ":"; m%; ":"; s%
ELSE
        PRINT "Time error"
END IF
```

```
Time= 10 : 39 : 54
```

CDBL　（式の値を倍精度定数型に変換）　関数

書式

CDBL(x)
　　　└─式

機能

xで示される式の値を，倍精度実数値に変換します．

CDBL$　（1バイト文字を2バイト文字に変換）　関数

書式

CDBL$(str)
　　　└─文字列

機能

strで示される文字列中の1バイト文字を，2バイト文字に変換します．

例

```
a$ = "Quick 123 !"
PRINT a$; "--->"; CDBL$(a$)
```

```
Quick 123 !--->Ｑｕｉｃｋ　１２３　！
```

CHAIN　　(プログラムのチェイン)

書式

CHAIN "filename"
　　　└─チェインするファイル

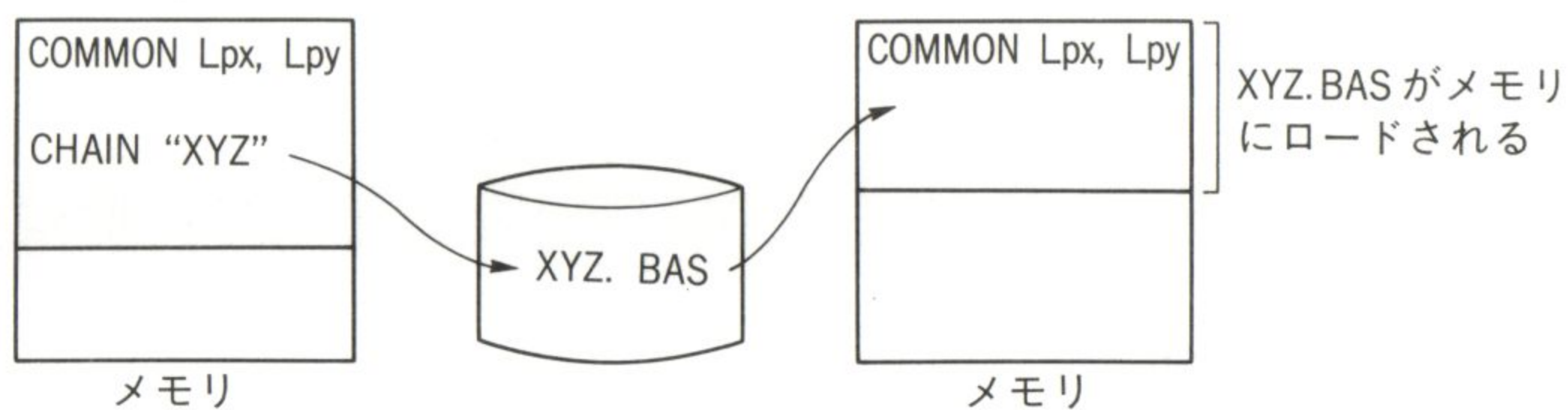
機能

現在実行中のプログラムをメモリ上から解放し，filenameで示されるファイルをロードし，制御を移します．

QB上でfilenameの拡張子を省略すると，.BASとみなされ，コマンドラインでの実行では.EXEとみなされます．

変数の値を，チェインするプログラムに引き渡すには，COMMON宣言しておきます．

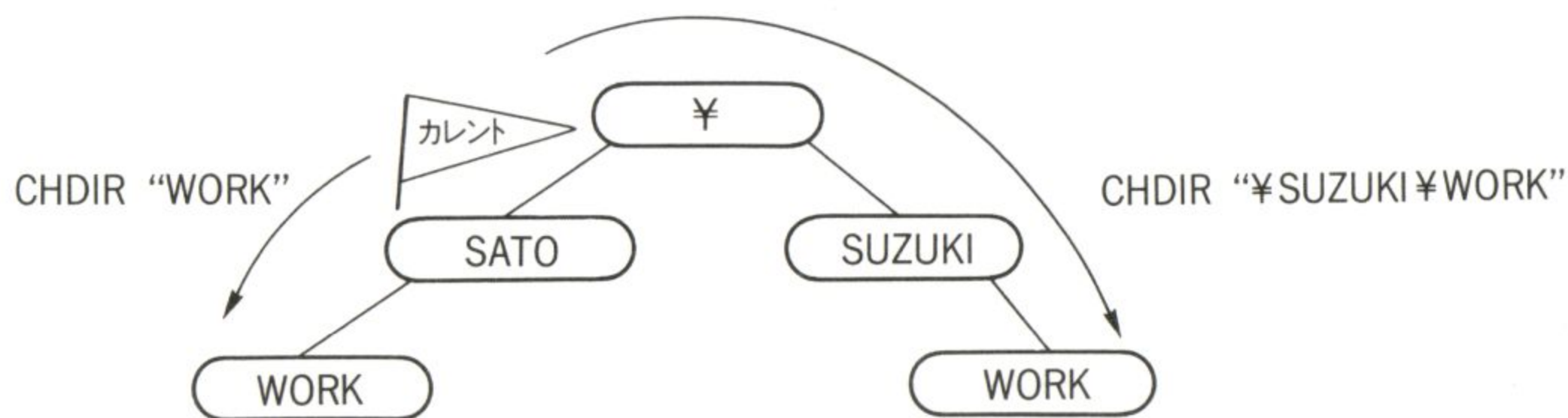
.EXEファイルでチェインする場合，BCOM42A.LIB(スタンドアロン型)はCOMMONをサポートしないので，COMMONを使用する場合は，BRUN42A.LIB(ランタイム分離型)を使用してください．

Ver.4.5ではBCOM45A.LIBとBRUN45A.LIBとなる．

CHDIR　　(カレント・ディレクトリの変更)

書式

CHDIR "path"
　　　└─パス名(サブディレクトリ名)

機能

カレント・ディレクトリを，pathで指定したサブディレクトリに移します．

CHR$　（ASCII コードを文字に変換）　関数

書　式

CHR$(code[,code…])
　　　↑ASCII コード

機　能

code で示される ASCII コード，または漢字連続コードを対応する文字に変換します．code を複数指定した場合は，それらの文字が連結されます．CHR$(65,1666)は“A 亜”を返します．漢字連続コードについては ASC を参照ください．

例

```
CLS
FOR k = 128 TO 255
    PRINT k; CHR$(k); "  ";
NEXT k
```

```
 128 ▁  129 ▂  130 ▃  131 ▄  132 ▅  133 ▆  134 ▇  135 █  136 ▏  137 ▎
 138 ▍  139 ▌  140 ▋  141 ▊  142 ▉  143 ┼  144 ┴  145 ┬  146 ┤  147 ├
 148 ▔  149 ─  150 │  151 ▕  152 ┌  153 ┐  154 └  155 ┘  156 ╭  157 ╮
 158 ╰  159 ╯  160    161 。  162 「  163 」  164 、  165 ・  166 ヲ  167 ァ
 168 ィ  169 ゥ  170 ェ  171 ォ  172 ャ  173 ュ  174 ョ  175 ッ  176 ー  177 ア
 178 イ  179 ウ  180 エ  181 オ  182 カ  183 キ  184 ク  185 ケ  186 コ  187 サ
 188 シ  189 ス  190 セ  191 ソ  192 タ  193 チ  194 ツ  195 テ  196 ト  197 ナ
 198 ニ  199 ヌ  200 ネ  201 ノ  202 ハ  203 ヒ  204 フ  205 ヘ  206 ホ  207 マ
 208 ミ  209 ム  210 メ  211 モ  212 ヤ  213 ユ  214 ヨ  215 ラ  216 リ  217 ル
 218 レ  219 ロ  220 ワ  221 ン  222 ゛  223 ゜  224 ═  225 ╞  226 ╪  227 ╡
 228 ◢  229 ◣  230 ◥  231 ◤  232 ♠  233 ♥  234 ♦  235 ♣  236 ●  237 ○
 238 ╱  239 ╲  240 ╳  241 円  242 年  243 月  244 日  245 時  246 分  247 秒
 248    249    250    251    252 ＼  253    254    255
```

CINT　（式の値を整数型に変換）　関数

書　式

CINT(x)
　　↑式

機　能

x で示される式の値を四捨五入して整数値に変換します．

例

INT 参照．

CIRCLE　　（円の描画）

書式

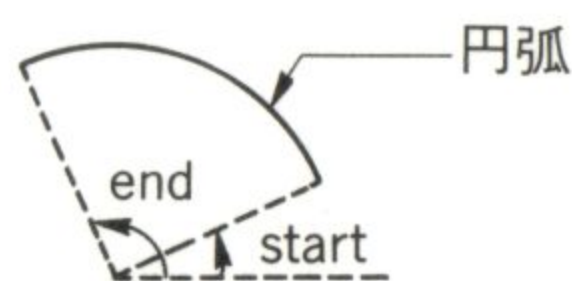
```
CIRCLE [STEP](x,y),r,[c],[start],[end],[aspect]
```

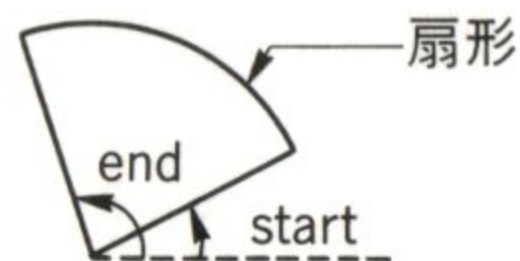
- (x,y)：中心座標
- r：半径
- c：色
- start：開始角
- end：終了角
- aspect：縦横比

機能

中心(x,y)で，半径rの円を色cで描きます．色cを省略すると，COLORで指定されている色となります．

startとendは，円の開始角と終了角でラジアンで指定します．

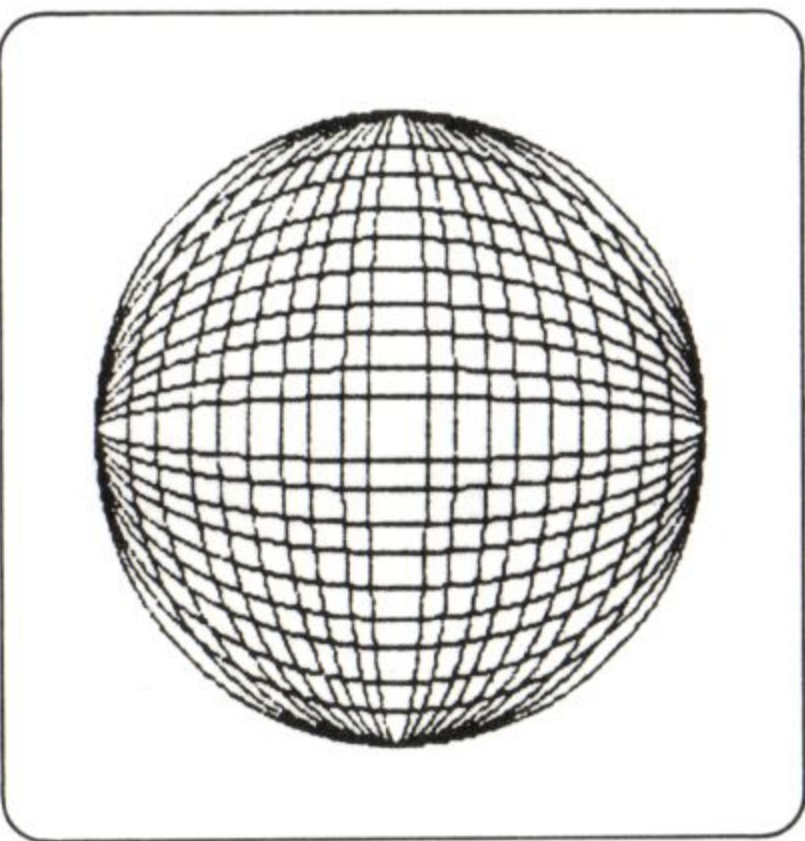
円弧
end
start

startとendに負の値を指定すると，絶対値をとった値で円弧を描き，中心と始点，中心と終点を結びます．つまり次のような扇形を描きます．

扇形
end
start

aspectは縦横比で，aspectが1より小さい場合は，x軸方向が長軸となり，1より大きい場合は，y軸方向が長軸になります．rの値が長軸となります．

(x,y)の前にSTEPを指定すると，(x,y)は絶対座標ではなく，現在位置(LP位置)からの相対座標となります．

CIRCLE文実行後の現在位置(LP位置)は円の中心に移ります．

例

```
' ---/* 楕円 */---

SCREEN 0: CLS
FOR k = 1 TO 10
    CIRCLE (320, 200), 100, , , , k / 10
    CIRCLE (320, 200), 100, , , , 10 / k
NEXT k
```

CLEAR　（変数の初期化）

書　式

```
CLEAR [, , stack]
            └─スタックサイズ
```

機　能

すべての変数をクリア(数値変数は0，文字列変数はヌル)し，すべてのファイルをクローズします．

stackを指定すると，〈デフォルトのスタックサイズ＋stack〉のサイズのスタックを確保します．

CLNG　（式の値を倍長整数型に変換）　関数

書　式

```
CLNG(x)
     └─式
```

機　能

式xの値の小数点以下を四捨五入して，倍長整数値に変換します．

CLOSE (ファイルのクローズ)

書式

CLOSE [[#]n ,…]
　　　　↑—**オープンされているファイル番号**

機能

ファイル番号nで示されるファイルをクローズします．nを省略するとオープンされているすべてのファイルをクローズします．

クローズにより，ファイル番号が解放され，出力バッファに残っているデータはファイルに書き出されます．

CLS (画面のクリア)

書式

CLS [{0 | 1 | 2}]
　　　↑—**クリア・モード**

機能

クリア・モードにしたがって画面をクリアします．

クリア・モード	機　能
0	すべてのテキスト画面とグラフィック画面をクリア
1	グラフィック画面のビューポート内をクリア
2	画面の最下行を残して，テキスト画面のビューポート内をクリア
指定なし	グラフィック画面とテキスト画面のビューポート内をクリア．画面の最下行にファンクションキー表示を復帰させる

COLOR (表示色の指定)

書式

COLOR [fore][,back]
　　　↑　　↑—**バックグラウンド・カラー**
　　　└—**フォアグラウンド・カラー**

機能

フォアグラウンド・カラーとバックグラウンド・カラーを，指定した色に設定します．PC-9801シリーズでは，テキスト画面に対するバックグラウンド・カラーは指定できません．

色の値は，テキスト画面とグラフィック画面とでは意味が異なります(PC-9801シリーズの場合)．

・グラフィック画面

色の値は，属性番号(アトリビュート)を指定します．

・テキスト画面

色の値は，色番号(カラーコード)を指定します．

属性番号と色番号は，PALETTE 文により対応付けられますが，SCREEN 文実行によりデフォルト値に対応付けられます．

グラフィック画面に表示できる色の種類は，SCREEN 文で指定するパレット・モード，およびコンピュータの機種(アナログ機／デジタル機)により異なります．

```
SCREEN   画面モード,パレット・モード
                      ├─1  …16色中16色．色番号0～15
                      ├─2  …64色中16色．色番号0～63
                      └─3  …4096色中16色．色番号0～4095
```

fore,back に指定できる属性番号および色番号は，次のとおりです．

		fore	back
アナログ機	テキスト	0～31(色番号)	―
	グラフィック	0～15(属性番号)	0～15(属性番号)
デジタル機	テキスト	0～31(色番号)	―
	グラフィック	0～7(属性番号)	0～7(属性番号)

テキストに指定する色番号と色の対応は，以下のとおりです．

色番号	0	1	2	3	4	5	6	7	8～15	16～31
色	黒	青	緑	水	赤	紫	黄	白	0～7の反転	0～15の点滅

グラフィック画面に指定する属性番号と色の対応は，PALETTE 文を参照してください．

COM(通信イベントのトラッピングの許可，禁止，停止)

書 式

```
COM (n) {ON | OFF | STOP}
      └─通信ポート番号(1または2)
```

機 能

nで示される通信ポートのイベント・トラッピングを次のように設定します．

ON …許可
OFF …禁止
STOP …停止(停止中に受信したイベントは保管されている)

COMMAND$（コマンド・ライン引数文字列の取得） 関数

書　式

COMMAND$

機　能

A>pro1␣abc␣xyz ↵
↓
COMMAND$

コマンド・ライン引数のプログラム名以後の文字列をCOMMAND$に取得します．コマンド・ラインの小文字は大文字に変換されて，COMMAND$に渡されます．

例

```
DIM Argv$(20)  ←引数を入れる文字配列

i = 1
FOR k = 1 TO LEN(COMMAND$)
    c$ = MID$(COMMAND$, k, 1)
    IF c$ <> " " THEN
        Argv$(i) = Argv$(i) + c$  ←Argv$(i)にi番目の引数が格納される
        sflag = 1  ←空白が続く場合はiを+1しないためのフラグ
    ELSEIF sflag = 1 THEN
        i = i + 1
        sflag = 0  ←
    END IF
NEXT k
Argc = i  ←引数の数をArgcに入れる

FOR k = 1 TO Argc
    PRINT Argv$(k)
NEXT k
```

このプログラムは，COMMAND$に取得したコマンド・ライン引数文字列を空白ごとにとり出し，Argv$(1)，Argv$(2)…に格納し，Argcにとり出した引数の数を格納するものです．

A>pro1␣abc␣xyz␣5 ↵
↓
COMMAND$
↓
Argv$(1) [abc]　Argc [3] ｝←引数の数
Argv$(2) [xyz]
Argv$(3) [5]

COMMON　　（共用変数の宣言）

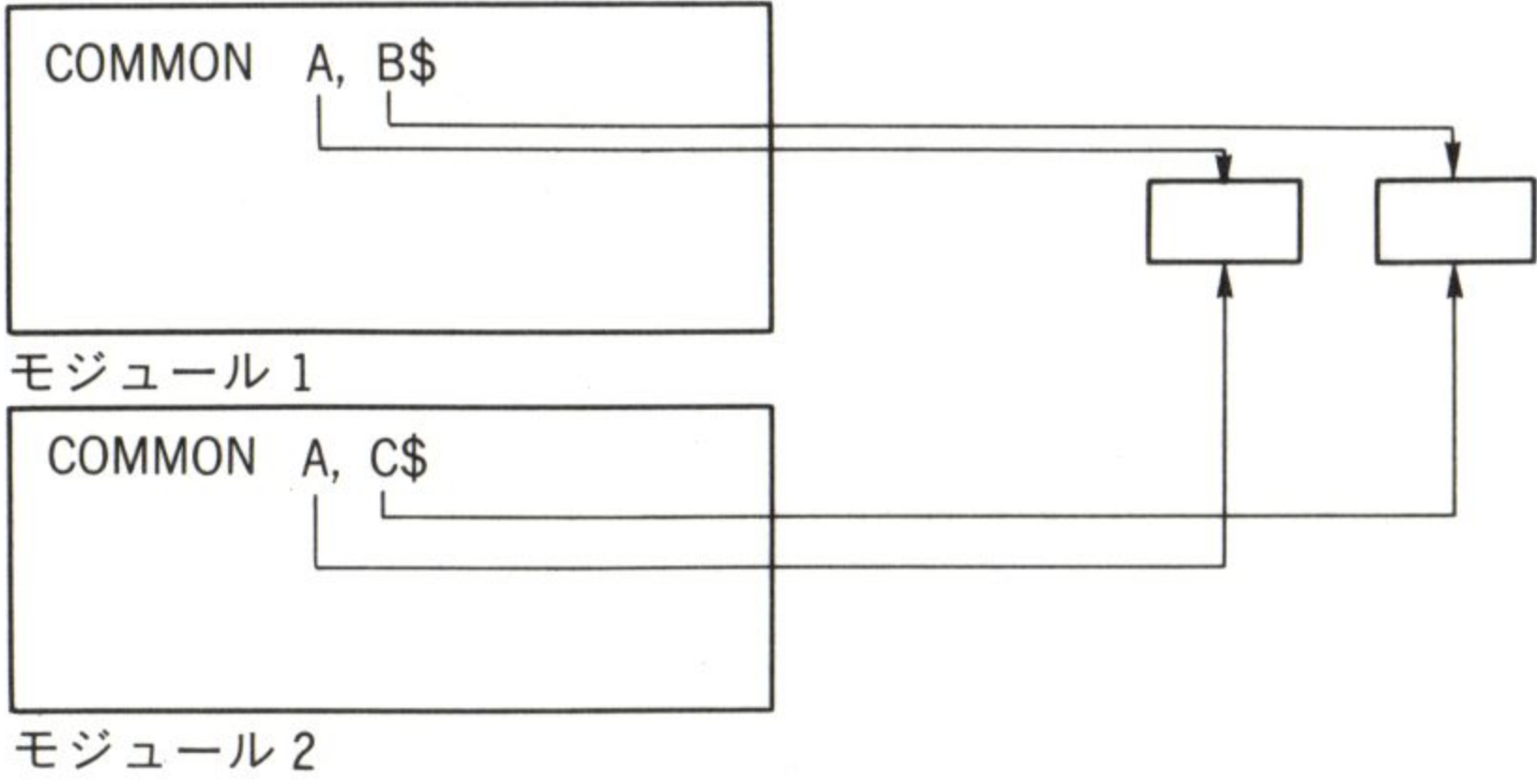
書　式

COMMON [SHARED] [/blockname/] var [AS type], var……

- /blockname/ — COMMON ブロック名
- var — 変数名
- type — 型名

機　能

var で示される変数を，複数のモジュール間で共用することを宣言します．

COMMON ステートメントで宣言された変数は，特別な場所に領域を確保し，そこを複数のモジュールが共用します．このため，COMMON によって宣言される変数は，名前ではなく，宣言された順序と型に依存します．

モジュール 1 の B$とモジュール 2 の C$は，変数名が異なりますが，共用されています．

COMMON A，B$と COMMON C$,A のように，2 つのモジュールで異なる型の順序で COMMON 宣言するとエラーとなります．

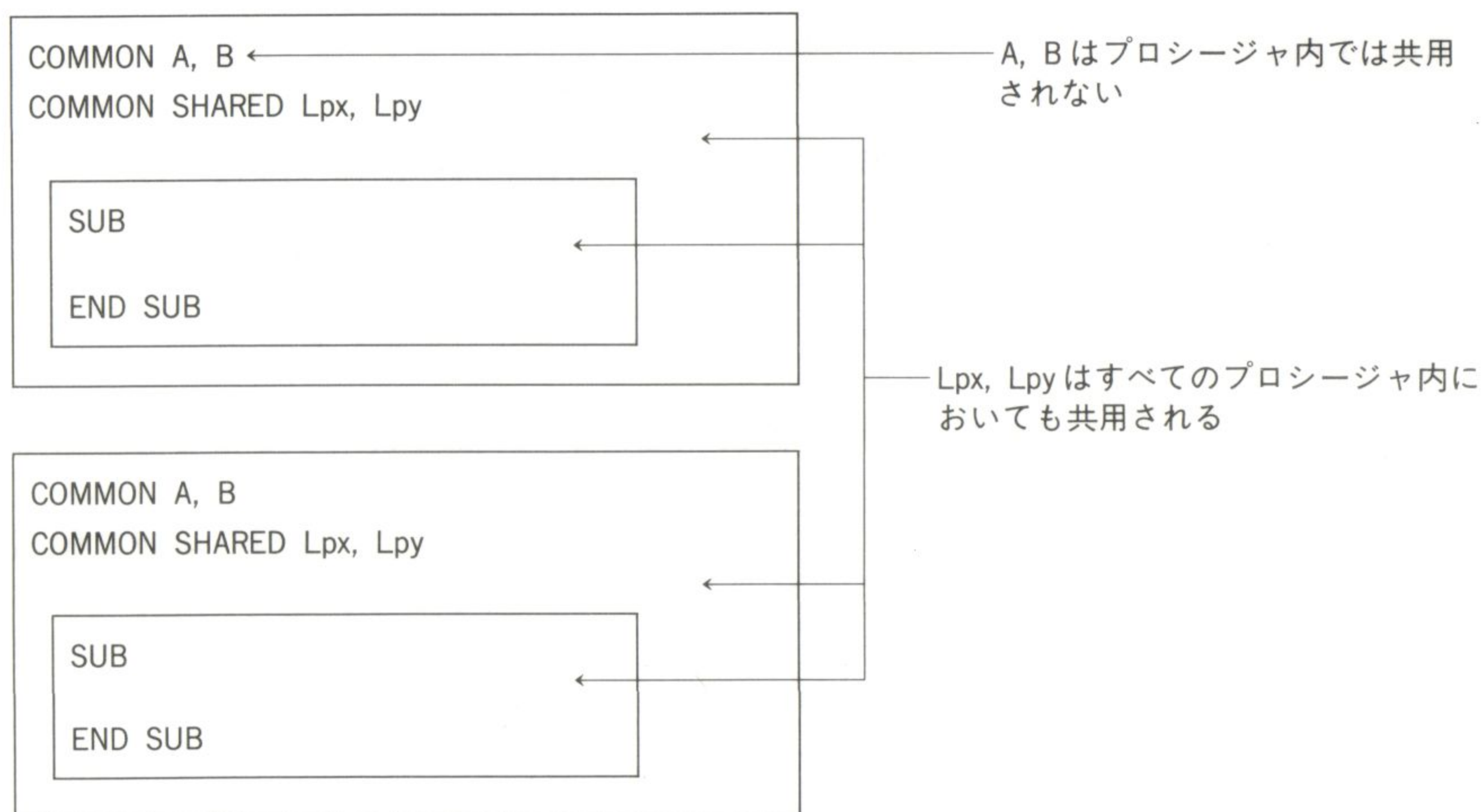
COMMON 文は，すべての実行ステートメントに先立って宣言する必要があります．実行ステートメントに属さないステートメントは以下のものです．

COMMON, CONST, DATA, DECLARE, DEFtyp, DIM, OPTION BASE, REM, SHARED, STATIC, TYPE～END TYPE, メタコマンド

● SHARED 属性

SHARED 属性を付けて COMMON 宣言すると，その変数は，モジュール内のすべてのプロシージャ間で共用されます．

```
COMMON A, B            ← A, B はプロシージャ内では共用されない
COMMON SHARED Lpx, Lpy
    SUB
    END SUB

COMMON A, B
COMMON SHARED Lpx, Lpy
    SUB
    END SUB
```

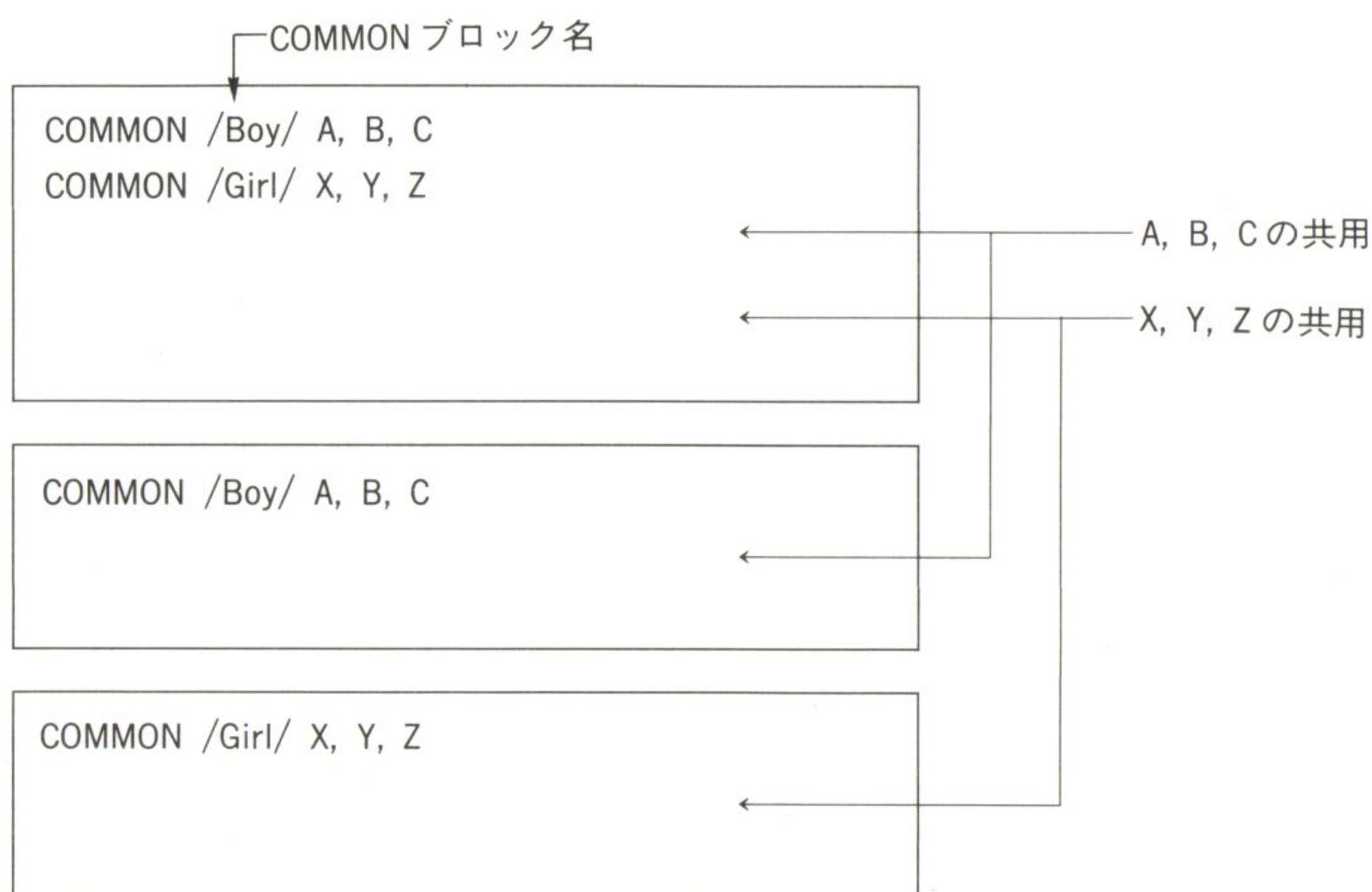
Lpx, Lpy はすべてのプロシージャ内においても共用される

● **COMMON ブロック名**

COMMON で宣言された変数は，宣言された順に共用領域に割り当てられていきますから，すべてのモジュールで同じ順序に宣言しなければなりません．

すべての変数を共用せずにある変数だけを共用するには，COMMON ブロック名を指定します．

```
   ┌COMMON ブロック名
COMMON /Boy/ A, B, C
COMMON /Girl/ X, Y, Z

COMMON /Boy/ A, B, C

COMMON /Girl/ X, Y, Z
```

A, B, C の共用

X, Y, Z の共用

COMMON ブロック名を指定したものを名前付き COMMON，COMMON ブロック名を指定しないものを空白の COMMON といいます．

名前付き COMMON 変数は，CHAIN で引き継げません．

●静的配列，動的配列の COMMON 宣言

静的配列の COMMON 宣言は，次のように COMMON 宣言の前に配列のサイズを指定します．

```
DIM   A(500),B(1000)
COMMON   A( ),B( )
```

動的配列の COMMON 宣言は，次のように COMMON 宣言の後で配列のサイズを指定します．

```
COMMON   A( ),B( )
DIM   A(500),B(1000)
```

CONST　（記号定数の定義）

書式

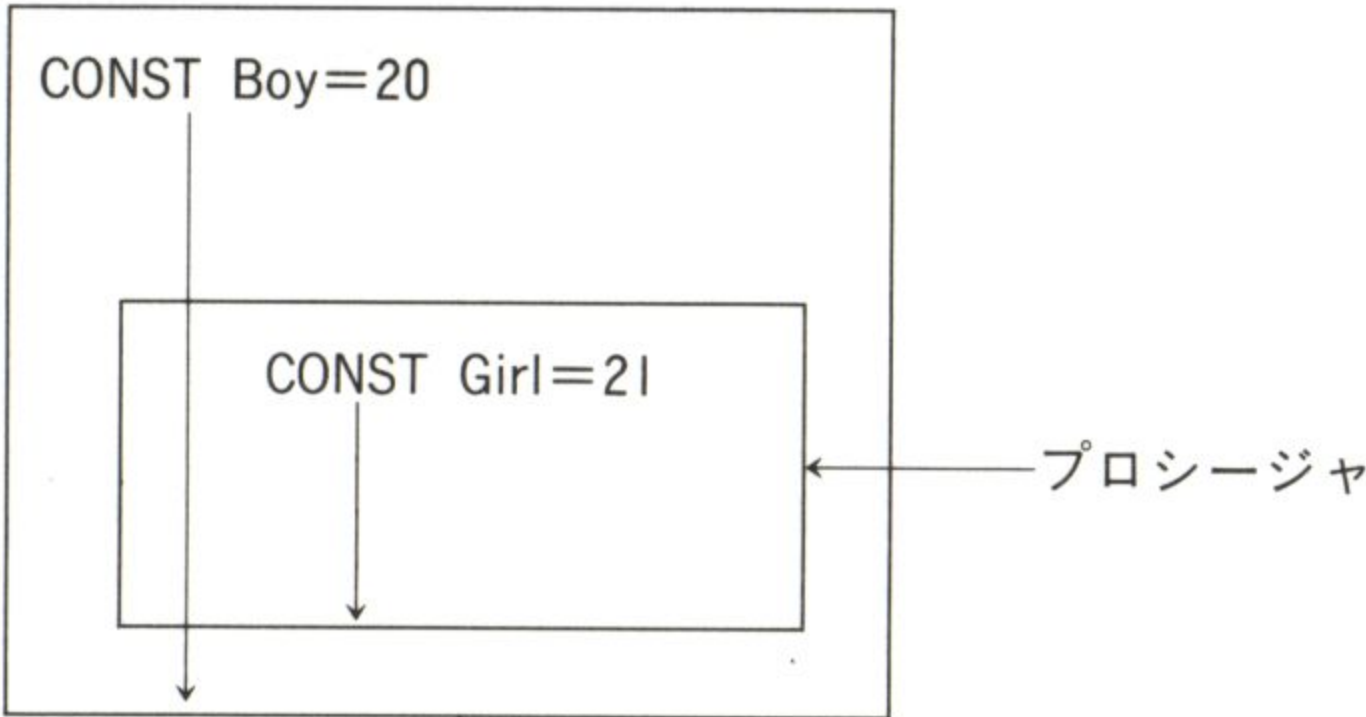
```
CONST   constname = expression, …
        ↑             ↑─式
        └─記号定数名
```

機能

constname で示される記号定数に expression で示す式の値を定義します．

式は，リテラル定数，記号定数，算術演算子(べき乗を示す^は含まない)，論理演算子から構成されるものです．式の中に変数や関数を含めることはできません．

記号定数を式の中に含める場合は，それが以前に定義されていなければなりません．

プロシージャ内で定義された記号定数は，そのプロシージャの中だけでローカルです．プロシージャ外で定義された記号定数は，モジュール全体でグローバルです．

記号定数名に型宣言文字（%，&，！，#，$）を付け加えて，型を示すこともできます．このように，型宣言文字を付加して定義された記

号定数を使用する際に，型宣言文字を省略すると，定義された型とみなされます．

例

```
CONST    Boy＝20，Girl＝21
CONST    Seito＝Boy＋Girl    ← Seito は20＋21＝41と定義される
   ⋮
DIM    Point(Seito)

CONST    MaxRec％＝100
   ⋮
DIM    A(MaxRec)
         ↑─整数型(％)として扱われる

CONST    Seito＝Boy＋Girl，Boy＝20，Girl＝21
          ↑─Boy と Girl が先に定義されていないのでエラー
```

COS （コサイン） 関数

書 式

COS(x)

機 能

x のコサインの値を返します．x の単位はラジアンです．

例

SIN 参照．

CSNG （式の値を単精度実数型に変換） 関数

書 式

CSNG(x)

機 能

式 x の値を単精度実数値に変換．

CSNG$ （2バイト文字を1バイト文字に変換） 関数

書 式

```
CSNG$(str)
       ↑─文字列
```

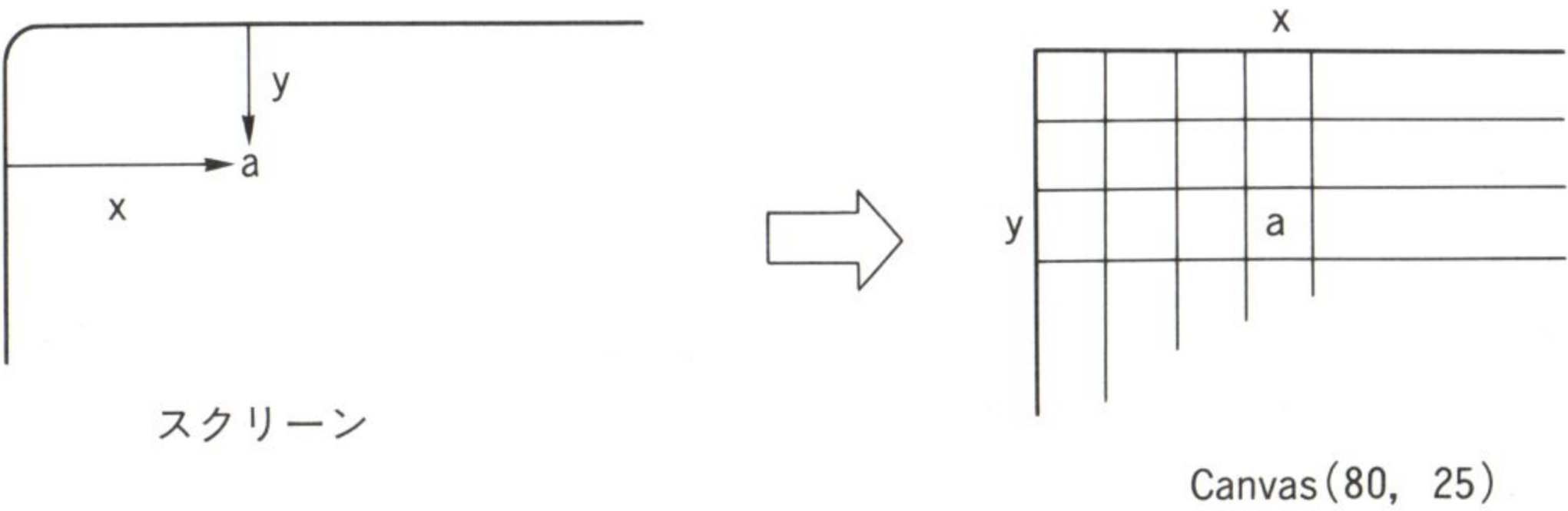
機 能

str で示される文字列中の2バイト文字を1バイト文字に変換します．CDBL$の逆の機能です．

CSRLIN　（現在のカーソル行の取得）　関数

書　式　CSRLIN

機　能　カーソルが現在位置している行を返します．

例　スクリーン上のデータを格納する配列Canvas(80,25)を用意しておき，スクリーン上の現在位置を取得して，それに対応した配列要素に，スクリーン上の文字を格納するものです．


```
DECLARE FUNCTION GetText! ()
DIM SHARED Canvas(80, 25) AS STRING *1

CLS
WHILE GetText <> -1
WEND

CLS
FOR y = 1 TO 25
    FOR x = 1 TO 80
        PRINT Canvas(x, y);
    NEXT x
NEXT y

FUNCTION GetText
    c$ = INKEY$
    IF c$ <> "" THEN PRINT c$;
    IF c$ >= " " THEN Canvas(POS(0), CSRLIN) = c$
    IF c$ = CHR$(27) THEN GetText = -1
END FUNCITON
```

配列に取得したデータの表示（FOR y ～ NEXT y）

現在位置の取得（POS(0), CSRLIN）

ESC キーがデータ入力の終わり（IF c$ = CHR$(27) THEN GetText = -1）

CVI/CVS/CVL/CVD (内部コード形式の文字列を実際の数値に変換)　関数

書式

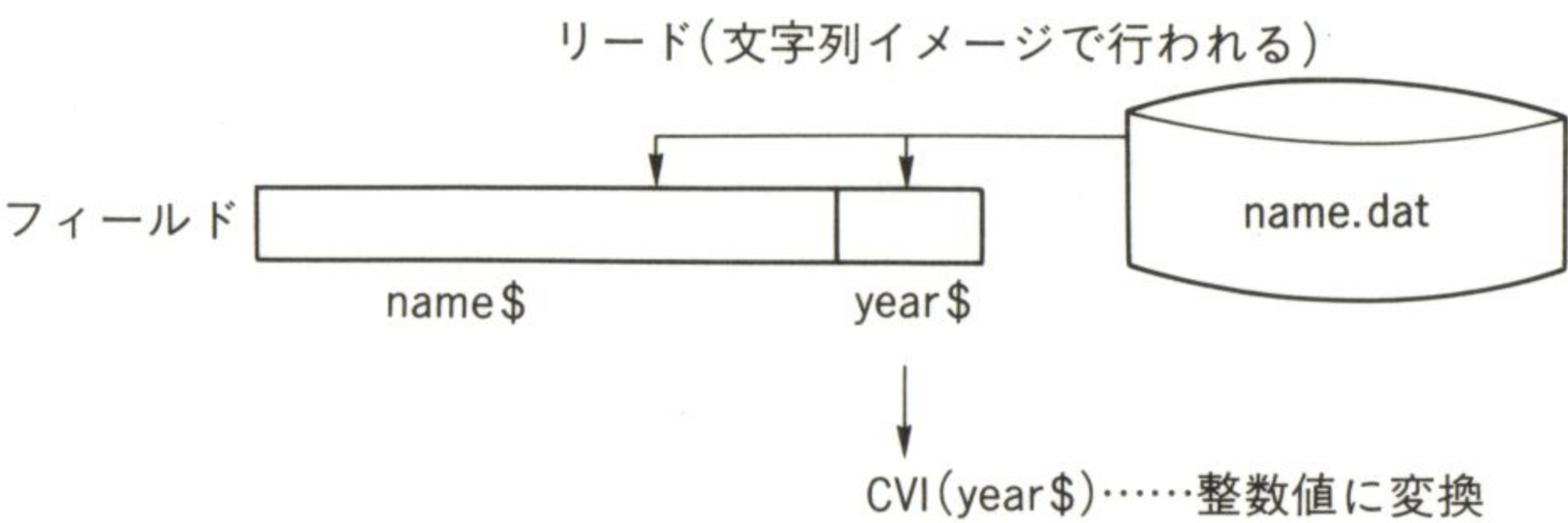
CVI(2-byte-string)　←　整数値に変換
CVS(4-byte-string)　←　単精度実数値に変換
CVL(4-byte-string)　←　倍長整数値に変換
CVD(8-byte-string)　←　倍精度実数値に変換

機能

ランダムアクセスファイルから，FIELDで定義した文字列変数にリードしたデータ(内部コード形式の文字列)を，実際の数値に変換します．

例

```
' ---/* ランダム・ファイルの読み込み */---

OPEN "b:name.dat" FOR RANDOM AS #1 LEN = 18
FIELD #1, 16 AS Name$, 2 AS Year$

INPUT "Rec No "; Rec
WHILE Rec > 0
    GET #1, Rec
    PRINT Name$; CVI(Year$) ←――Year$のデータを整数値に変換
    INPUT "Rec No "; Rec
WEND
CLOSE #1
```

```
Rec No? 1
Candy          21
Rec No? 4
Rolla          20
Rec No? 0
```


CVSMBF/CVDMBF (MBF 文字列をIEEE形式の数値に変換)　関数

書式

CVSMBF(4-byte-string)　←　IEEE 形式の単精度実数値に変換
CVDMBF(8-byte-string)　←　IEEE 形式の倍精度実数値に変換
　　　　↑MBF 文字列

機能

マイクロソフト・バイナリ形式(MBF)の文字列を，IEEE 形式の数値に変換します．

DATA （リードするデータの定義）

書式

DATA constant1, …
↑定数

機能

READ文で読むデータを定義します．定数にはリテラル定数を指定します．記号定数を指定した場合，それは記号定数としてではなく単なる文字列としてリードされます．

文字列は二重引用符(″)で囲む必要はありませんが，カンマやコロンを文字列中に含める場合は，二重引用符で囲んでください．

データの並びの中には，ヌルデータ項目(カンマだけで区切られていてデータがないもの)を置くことができ，その項目のデータは，数値変数に読まれれば0，文字列変数に読まれればヌル(″″)となります．

例

DATA 10,Hello,,20，日本語，″a,b ″
↑ヌルデータ項目

DATE$ （現在日付の取得） 関数

書式

DATE$

機能

現在日付を，yyyy-mm-ddの10バイト文字列として取得します．

yyyy： 年（1980～2099）
mm： 月（01～12）
dd： 日（01～31）

例

```
PRINT "現在の日付-->"; DATE$
INPUT "年,月,日"; year$, month$, day$
DATE$ = year$ + "-" + month$ + "-" + day$
```

```
現在の日付-->1990-03-31
年,月,日? 1990,4,1
```

DATE$　（日付の設定）

書式

```
DATE$ = str
        ↑日付を示す文字列
```

機能

str で示す日付に現在日付を設定します．str は，yyyy-mm-dd または yy-mm-dd の文字列です．

yyyy：年（1980～2099）
yy：年（00～99）
mm：月（01～12）
dd：日（01～31）

日付は，-(ハイフン)の代わりに／(スラッシュ)で区切ってもかまいません．

DECLARE　（プロシージャの宣言）

書式

```
DECLARE {FUNCTION | SUB } name (var [AS type], …)
```

name ← プロシージャ名
var ← 引数
type ← 引数の型

機能

プロシージャの使用に先立って，プロシージャ名と引数の型を宣言します．

QB 統合環境において，プログラムをセーブすると，SUB プロシージャ，FUNCTION プロシージャ定義に対応した，DECLARE ステートメントが自動的に生成されます．

例

```
DECLARE  SUB  Disp(D$, Num)←プロシージャの宣言
   ⋮
   Disp  "Hello", 5←プロシージャの呼び出し
   ⋮
SUB  Disp(D$, Num)
 ⋮                        ←プロシージャの定義
END  SUB
```

DECLARE (他言語プロシージャの宣言)

書式

```
                              ┌─プロシージャ名            ┌─オブジェクト中の別名
DECLARE {FUNCTION | SUB } name [CDECL] [ALIAS "aliasname "]
                  [{BYVAL | SEG }] arg [AS type],…
                                    ↑          ↑─引数の型
                                    └─引数
```

機能

BASIC以外の言語で書かれた外部プロシージャのプロシージャ名と引数の型の宣言をします.

CDECL … プロシージャがC言語のコール手順を使用することを意味します. CDECLを指定すると, 引数は右のものからスタックに積まれ, プロシージャ名nameは小文字に変換されて, アンダーバーが先頭に付加されます.

ALIAS … プロシージャがオブジェクト・ファイル(ライブラリ・ファイル)中で, 別の名前で登録されていることを示します. 別名をaliasnameで指定します.

BYVAL… 引数が参照(デフォルト)によってではなく, 値によって渡されることを意味します.

SEG … 引数がfar参照で渡されることを意味します.

DEF FN (関数の定義)

書式

```
① DEF FNname[(arg [AS type],…)]=expression
         ↑     ↑         ↑─引数の型    ↑─式
      関数名   └─引数
② DEF FNname[(arg [AS type],…)]
   :   ↑─関数名 ↑─引数   ↑─引数の型
     FNname=expression
   :           ↑─式
  END DEF
```

機能

①の書式は1論理行に収まる範囲で関数を定義でき, ②の書式は複数行に渡って関数を定義できます.

関数名の先頭には必ずFNが付きます. 関数の定義は呼び出しの前に行われていなければなりません.

DEF FN関数は再帰的(自分自身の定義内から自分自身を呼び出す)に定義できません.

DEF FN関数内の変数はグローバル変数です. ローカル変数にしたい場合は, その変数をSTATIC宣言してください.

例

```
' ---/* 関数の定義 */---

DEF FNHeihouwa (a, b) = a * a + b * b   ←――― a²+b²を求める関数

DEF FNMax (a, b)                         ┐
    IF a > b THEN                        │
        FNMax = a                        │
    ELSE                                 ├←――― aとbの
        FNMax = b                        │     大きい方を返す関数
    END IF                               │
END DEF                                  ┘

PRINT FNHeihouwa(3, 4)
PRINT FNMax(10, 20)
```

DEF SEG　(セグメント・アドレスの設定)

書式

DEF SEG [= address]
　　　　　　↑― **セグメント・アドレス**

機能

セグメントの先頭アドレスを，address で示されるセグメント値に設定します．BLOAD，BSAVE，CALL ABSOLUTE，PEEK，POKE で指定するオフセットは，このセグメント・アドレスからの距離です．

address を省略すると，BASIC のデータセグメントが採用されます．

例

BSAVE 参照.

DEFtyp　(変数名,関数名のインプリシット宣言)

書式

DEFtyp　letter1[- letter2], …
↑　　　　　　↑― **文字の範囲**
└― **型**

機能

変数名，関数名(DEF FN 関数，FUNCTION プロシージャ)の先頭文字が，letter1～letter2 で示される文字であれば，その変数名，関数名の型を type としてみなすように宣言します．

type として次の 5 つがあります．

INT	…整数型	LNG	…倍長整数型
SNG	…単精度実数型	STR	…文字列型
DBL	…倍精度実数型		

型宣言文字(%, &, !, #, $)は，DEFtyp 文に対してつねに優先します．

DEFtyp 文は，レコードのメンバ名には効果を持ちません．

例

```
DEFINT    A-Z              …すべての変数名，関数名の型は整数型となる．

DEFDBL    A, S, X-Z        …A, S, X-Zで始まる変数名，関数名の型は
   ⋮                         倍精度実数型となる．
  Sum ←倍精度実数型
  Small %←整数型
```

DIM　（変数，配列の宣言）

書式

```
DIM [SHARED] var [([lower TO]upper,…)] [AS type],…
```

- var ― 変数，配列
- lower ― 添字の下限値
- upper ― 添字の上限値
- type ― 変数の型

機能

変数の型宣言，配列のサイズと型の宣言をします．

配列の添字は，TOを用いて下限値と上限値を指定することもできます．指定できる添字の値は−32768～32767です．

typeで指定できる型は，INTEGER，LONG，SINGLE，DOUBLE，STRING(可変，固定長)があります．

SHAREDを指定すると，その変数，配列はモジュール内のすべてのプロシージャで使用できます．

例

```
DIM   A(−5  TO  5, 10  TO  20)      …配列A( )の宣言
```

	10	…	20
−5			
⋮			
5			

```
DIM  Sum, Num  AS  INTEGER
                 …変数Sumは単精度実数型，変数Numは整数型．2つとも整数型と
                   するには，DIM  Sum  AS  INTEGER, Num  AS  INTEGERと宣言
                   する．

TYPE   Eisei
  Star  AS  STRING *10
  Kodo  AS  INTEGER
  Shuki  AS  SINGLE
END   TYPE
DIM A AS Eisei       …AはEisei型の変数
```

DO～LOOP （くり返し）

書　式

① DO
　⋮
LOOP {WHILE | UNTIL } booleanexpression
　　　　　　　　　　　　　　　　↑─条件式

② DO {WHILE | UNTIL } booleanexpression
　⋮　　　　　　　　　　　　　　　↑─条件式
LOOP

機　能

①ループの後部でくり返し条件を判定する後判定反復
②ループの前部でくり返し条件を判定する前判定反復
WHILE は条件式が真である間ループをくり返し,UNTIL は条件式が真になるまでループをくり返します.
WHILE NOT booleanexpression と UNTIL booleanexpression およびこの逆は同じことを表します.

例

```
' ---/* 繰り返し */---

DIM Name$(100)
DATA Candy,Eluza,Nina,Lisa,Rolla,/

Num = 1
READ Name$(Num)
DO WHILE Name$(Num) <> "/"         …Name$()にデータをリードする.
    Num = Num + 1                   WHILE 型の前判定反復.
    READ Name$(Num)
LOOP
Num = Num - 1



Num = 0
DO
    Num = Num + 1                   …上と同じことを WHILE 型の後判定反復
    READ Name$(Num)                  で書いたもの.
LOOP WHILE Name$(Num) <> "/"
Num = Num - 1



Num = 0
DO
    Num = Num + 1                   …上と同じことを UNTIL 型の
    READ Name$(Num)                  後判定反復で書いたもの.
LOOP UNTIL Name$(Num) = "/"
Num = Num - 1
```

DRAW　　　　（マクロコマンドによる描画）

書　式

DRAW stringexpression
　　　　↑マクロコマンドを示す文字列式

機　能

stringexpressionで示されるマクロコマンドにしたがった描画をします．

マクロコマンドは以下のとおりです．

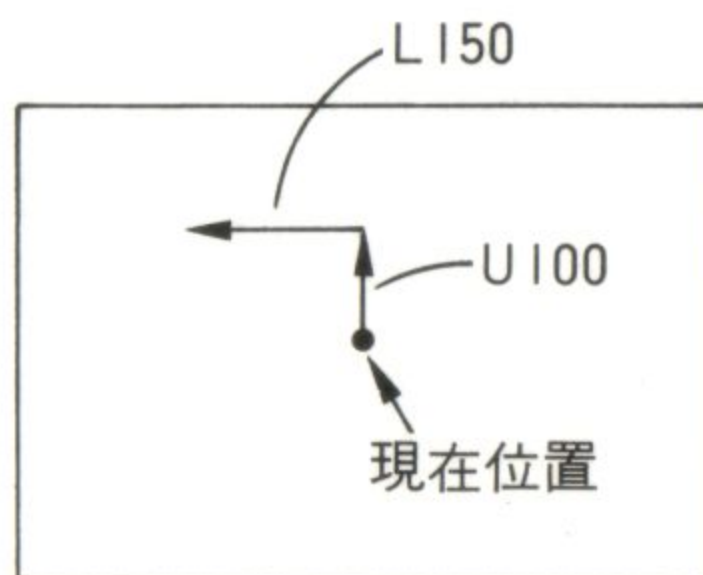
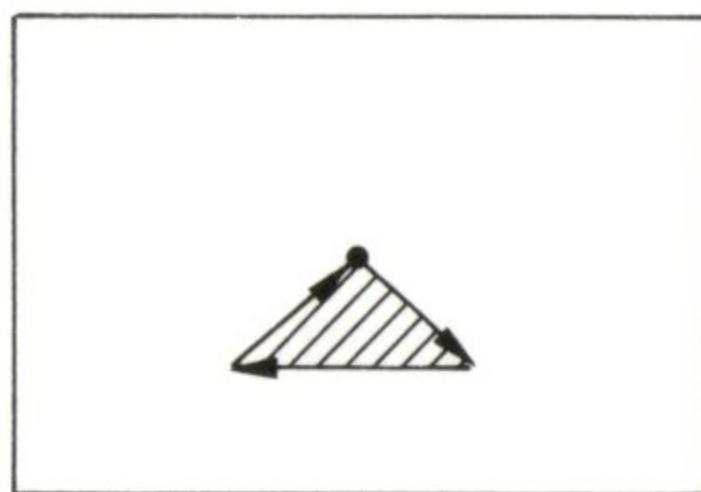
プレフィックス	B N	線を引かずに移動 移動命令の位置まで線を引き，その後(0, 0)の位置に戻る
移動	U [n] D [n] L [n] R [n] E [n] F [n] G [n] H [n] Mx,y	n単位分，上に移動 n単位分，下に移動 n単位分，左に移動 n単位分，右に移動 n単位分上，n単位分右に斜めに移動 n単位分下，n単位分右に斜めに移動 n単位分下，n単位分左に斜めに移動 n単位分上，n単位分左に斜めに移動 (x, y)位置に移動．x, yの前に＋または－の符号を付けると，現在位置からの相対移動
回転・色・倍率	An TAn Cn Sn Pn,m	以後描く図形をnで示す角度(0：0°, 1：90°, 2：180°, 3：270°)だけ回転 以後描く図形をn°回転．nは－360～360 以後描く図形の色をnで示す色に設定 以後描く図形の拡大倍率をn倍に設定．nは1～255の値である．Mコマンドは相対移動のときだけ倍率がかけられ，絶対移動では倍率はかけられない 現在位置を中心に，色mで枠どられた図形の中を色nでペイント(ぬりつぶし)する
その他	"X"＋ VARPTR$(str) "command=" +VARPTR$(num)	文字列変数strに格納されているマクロコマンドを実行 commandで示すマクロコマンドの引数として，数値式numで示される値を用いる

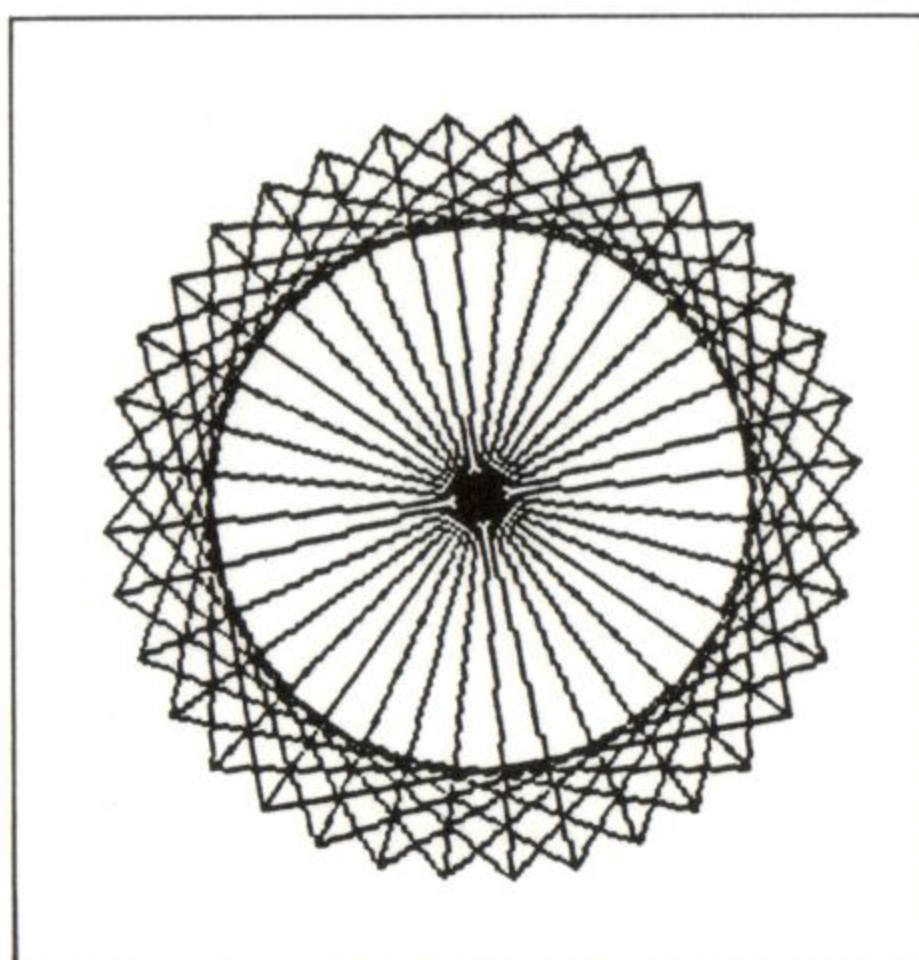
プレフィックスB,Nは，次に来る移動コマンド(U～M)に対してだけ作用します．

プログラム開始時の描画現在位置は，画面の中央(320,200)です．

例

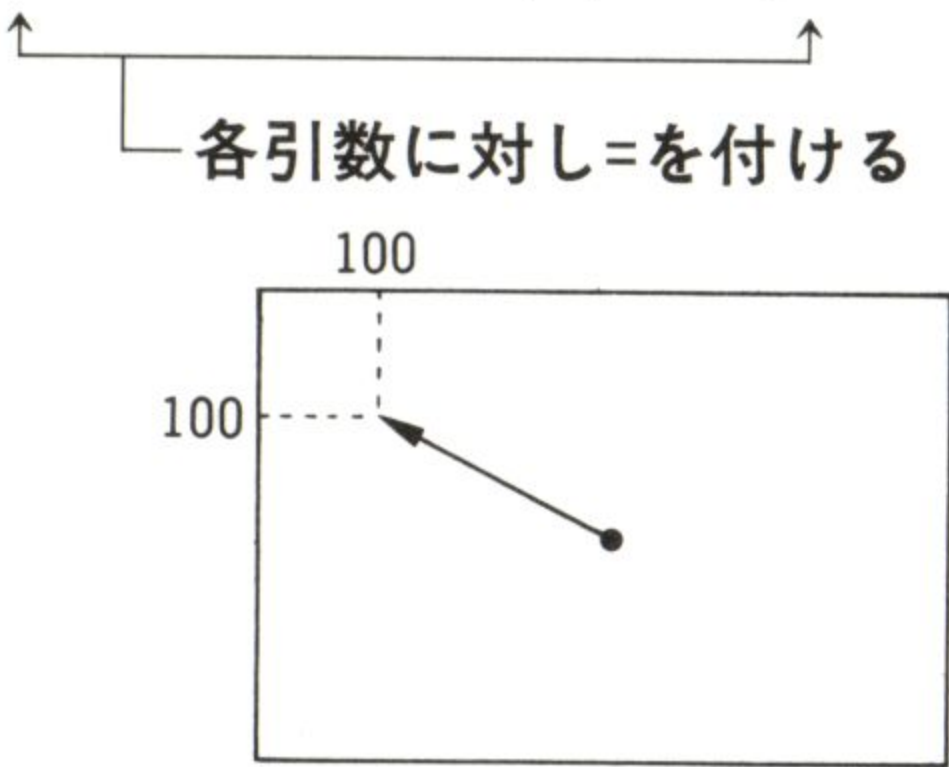
```
DRAW ″U100 L150 ″
```

L150
U100
現在位置

```
DRAW ″C2″           ←色をマゼンダ
DRAW ″F60L120E60″   ←三角形を描く
DRAW ″BD60″         ←三角形の内部に移動
DRAW ″P1,2″         ←内部をペイント
```

```
FOR A=0 TO 360 STEP 10                  ┌─三角形の描画
   DRAW ″TA=″+ VARPTR$(A)+ ″F60L120E60″
NEXT A
```

…△の三角形を10°ごとに回転して描く.

```
A =100 : B =100
DRAW "M =" + VARPTR$(A) + ",=" + VARPTR$(B)
```

各引数に対し=を付ける

100

100

END　（プログラム，プロシージャの終了）

書　式

① END
② END DEF
③ END FUNCTION
④ END IF
⑤ END SELECT
⑥ END SUB
⑦ END TYPE

機　能

①プログラムの終了．この位置でプログラムの実行を停止し，OS または QB 統合環境に制御を戻します．
② DEF FN 関数の定義の終わり
③ FUNCTION プロシージャの定義の終わり
④ブロック IF 文の終わり
⑤ SELECT CASE 文の終わり
⑥ SUB プロシージャの定義の終わり
⑦ユーザ定義型の定義の終わり

ENVIRON$　（DOS 環境文字列の取得）　関数

書　式

① ENVIRON$(str)　← 環境変数名
② ENVIRON$(n)　← 環境文字列テーブル内の番号

機　能

① str で示される DOS 環境変数名に割り当てられている文字列を取得します．
②環境文字列テーブル内の n 番の環境変数名と，それに割り当てられている文字列を取得します．

環境変数が見つからなければヌル文字列を返します．文字列の大/小文字は区別されます．
つまり LIB と lib は異なる名前とみなされます．

ENVIRON$(LIB) …A : ¥LIB
ENVIRON$(2) …LIB = A : ¥LIB

1	PATH＝A : ¥BIN ; A : ¥
2	LIB＝A : ¥LIB
3	
⋮	⋮

環境文字列テーブル

例

```
' ---/* ＤＯＳ環境文字列の取得 */---

n = 1
DO WHILE ENVIRON$(n) <> ""
    PRINT n; " : "; ENVIRON$(n)
    n = n + 1
LOOP
```

…環境文字列テーブルに登録されている文字列を表示

```
1  : COMSPEC=A:¥COMMAND.COM
2  : PATH=A:¥BIN
3  : LIB=A:¥LIB
```

ENVIRON （DOS 環境文字列の設定）

書　式

ENVIRON ″parameter = text″

- parameter ← 環境変数名
- text ← 文字列

機　能

parameter で示される環境変数に，text で示される文字列を割り当てます．この環境変数は，環境文字列テーブルの末尾に付け加えられます．もし環境変数がすでにテーブル内にあれば削除されます．
text がヌル文字列 (″″) またはセミコロン(″；″)なら，環境文字列テーブルから環境変数が削除されます．
parameter = text は parameter text と書くこともできます．

例

ENVIRON ″PATH = A : ¥BIN ; A : ¥ ″
…PATH 環境変数を A : ¥BIN ; A : ¥に設定

EOF　（ファイル終わりの検知）　関数

書式

```
EOF(n)
   └─ファイル番号
```

機能

ファイル番号nでオープンされているファイルの，ファイルエンドで真(－1)を返します．

EOFはデバイス・ファイルSCRN:,KYBD:,CONS:,LPTn:には使用できません．

通信デバイスをアスキーモードでオープンしている場合，CTRL＋Zの受信でEOFは真(－1)となり，デバイスをクローズするまで，真の状態を保ちます．バイナリモードでオープンしている場合は，入力待ち行列が空(LOC(n)=0)であればEOFは真となり，空でなければEOFは偽となります．

例

INPUT$,INPUT#,LINE INPUT# 参照．

ERASE　（配列変数の初期化と解放）

書式

```
ERASE arrayname,…
      └─配列名
```

機能

①静的配列の場合

指定した配列が静的配列の場合，配列要素は，数値配列なら0，文字列配列ならヌル文字列に初期化されます．

②動的配列の場合

指定した配列が動的配列の場合，その配列が割り当てられているメモリが解放されます．

例

```
REM   $DYNAMIC
DIM   A(100,100)
  ⋮
ERASE   A      ←配列A( )が割り当てられていたメモリを解放

REM   $STATIC
DIM   B(100,100)
  ⋮
ERASE   B      ←配列B( )のすべての要素を0に初期化
```

ERDEV　（デバイスのエラーコードの取得）　関数

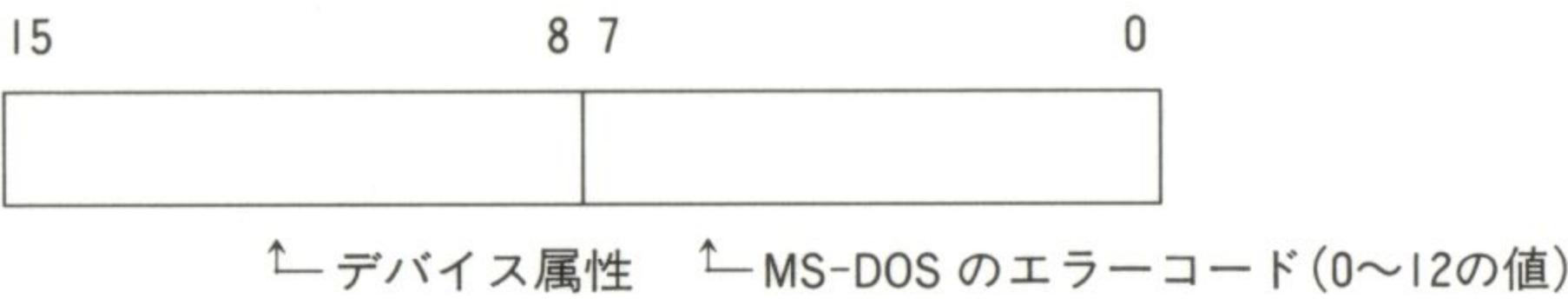
書式　ERDEV

機能

エラーが発生したデバイスからのエラーコードを返します．この関数は通常，ON ERROR 文で指定した，エラー処理ルーチン内で使用します．

エラーコードのビットの意味は次のとおりです．

15　　　　8	7　　　　0
↑デバイス属性	↑MS-DOS のエラーコード(0〜12の値)

例

```
' ---/* エラー・ハンドラ */---

ON ERROR GOTO ErrorHandler

OPEN "b:abc.qat" FOR INPUT AS #1


END

ErrorHandler:
    PRINT "エラーのあったデバイス  : "; ERDEV$
    PRINT "ＭＳ－ＤＯＳエラーコード: "; ERDEV
RESUME NEXT
```

```
エラーのあったデバイス  :  B:
ＭＳ－ＤＯＳエラーコード:   2
```

ERDEV$　（エラーが発生したデバイス名の取得）　関数

書式　ERDEV$

機能

エラーが発生したデバイスのデバイス名を返します．この関数は通常 ON ERROR 文で指定した，エラー処理ルーチン内で使用します．

例

ERDEV 参照．

ERL　（エラーがあった行番号の取得）　関数

書式

ERL

機能

エラーが発生した行の行番号を返します．なお，その行には行番号を付けておく必要があります．この関数は通常，ON ERROR 文で指定した，エラー処理ルーチン内で使用します．

例

ERR 参照．

ERR　（エラーコードの取得）　関数

書式

ERR

機能

発生したエラーのエラーコードを返します．この関数は通常，ON ERROR 文で指定した，エラー処理ルーチン内で使用します．

例

```
' ---/* エラー・ハンドラ */---

ON ERROR GOTO ErrorHandler

100 a = LOG(-1)
110 b = 10 / 0
 ↑―――――――――――行番号
END


ErrorHandler:
    ErrorNum = ERR      ←――――――エラーコード
    ErrorLine = ERL     ←――――――エラーのあった行番号
    PRINT "Error in"; ErrorLine; " : ";
    SELECT CASE ErrorNum
        CASE 5
            PRINT "引数が許される範囲ではありません"
        CASE 6
            PRINT "計算結果がオーバーフローしました"
        CASE 7
            PRINT "メモリが足りません"
        CASE 9
            PRINT "配列の添字が許される範囲にありません"
        CASE 11
            PRINT "0で除算しました"
        CASE ELSE
            PRINT "その他のエラー"
    END SELECT
RESUME NEXT
```

```
Error in 100  : 引数が許される範囲ではありません
Error in 110  : 0で除算しました
```

ERROR (エラーコードに対応したエラーメッセージの表示)

書　式

```
ERROR n
      └─エラーコード
```

機　能

エラーコードn(0～255)に対応した，エラーメッセージを表示します．

例

```
ERROR 1
```

画面上に以下のようなエラーメッセージが表示され，ERROR 文のところで実行を止める．

対応するFOR がありません(ERR =1)

確認

EXIT (ループ，プロシージャからの脱出)

書　式

① EXIT DEF
② EXIT DO
③ EXIT FOR
④ EXIT FUNCTION
⑤ EXIT SUB

機　能

① DEF FN 関数から抜けます．
② DO～LOOP から抜けます．ループが入れ子になっている場合，そのループのすぐ外側のループに出ます．
③ FOR～NEXT から抜けます．ループが入れ子になっている場合，そのループのすぐ外側のループに出ます．
④ FUNCTION プロシージャから抜けます．
⑤ SUB プロシージャから抜けます．

例

```
' ---/* ループからの脱出 */---

KeyDat = 5

FOR k = 1 TO 10
    READ a
    IF a = KeyDat THEN
        PRINT k, a
        EXIT FOR
    END IF
NEXT k

DATA 2,3,1,5,6,0,1,2,1,1
```

…10個のデータを先頭から調べ，KeyDat と同じデータが最初に見つかったときにループから抜ける．

```
DECLARE FUNCTION Search! (a!(), KeyDat!, n!)
' ---/* 関数からの脱出 */---

DIM a(10)
FOR k = 1 TO 10
    READ a(k)
NEXT k
DATA 2,3,1,5,6,0,1,2,1,1

PRINT Search(a(), 15, 10)
PRINT Search(a(), 6, 10)


FUNCTION Search (a(), KeyDat, n)
    FOR k = 1 TO n
        IF a(k) = KeyDat THEN
            Search = k
            EXIT FUNCTION ←―――――ファンクションプロシージャ Search
        END IF                   から呼び出し元に戻る
    NEXT k
    Search = -1 ←―――――――データが見つからなかったとき
END FUNCTION
```

…Search は配列 a()の中から KeyDat を探し，
その位置を返す．

EXP　（指数）　関数

書　式　EXP(x)

機　能　指数 e^xの値を返します．x には88.02969以下の値を与えてください．

FIELD　（ファイルバッファの設定）

書　式

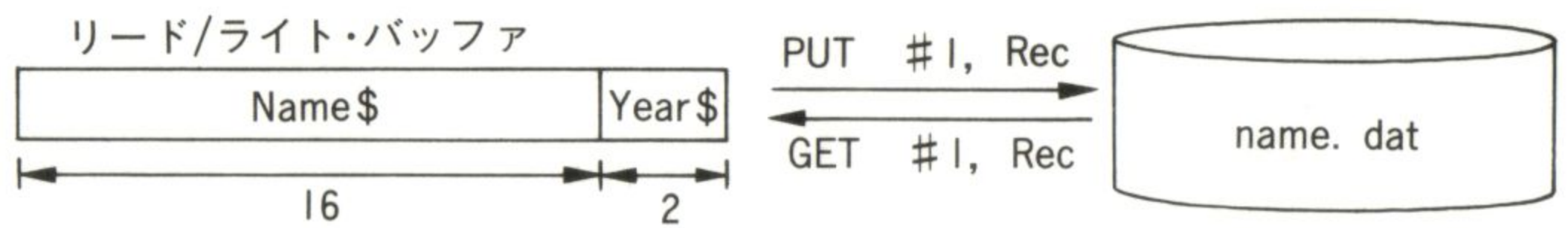
```
FIELD  [#]n ,w AS strvar, …
        ↑   ↑         ↑
        │   │         └文字列変数
        │   └strvar に入れる文字列の長さ
        └ファイル番号
```

機　能　ランダム・ファイルとの間のリード/ライト・バッファの設定を行います．

```
OPEN  "b:name.dat" FOR RANDOM AS #1 LEN=18
FIELD #1,16 AS Name$, 2 AS Year$
```

リード/ライト・バッファ

Name$	Year$
16	2

PUT #1, Rec →
← GET #1, Rec
name. dat

上の例では，Name$，Year$という2つのフィールドからなる18バイトのリード/ライト・バッファを設定しています．GET,PUTは，このバッファとファイルとの間でリード/ライトを行います．

同じファイルに対して，FIELD文で異なるバッファを設定することができ，これらは同時に有効です．

バッファのフィールドとして宣言されている変数(上の例のName$，Year$)に対して，INPUTや代入文で値を代入しないでください．

例　CVI,MKD$参照．

FILEATTR　（ファイル属性の取得）　関数

書式

```
FILEATTR(n,flag)
          ↑  └─取得情報の種類
          └─ファイル番号
```

機能

flag =1のときは，ファイル番号nのファイルのモードを次の値で返します．

1	INPUT
2	OUTPUT
4	RANDOM
8	APPEND
32	BINARY

flag =2のときは，ファイル番号nのファイルのMS-DOS上のファイルハンドルを返します．

例

```
OPEN "LPT1:" FOR OUTPUT AS #1
OPEN "abc.dat" FOR INPUT AS #2

PRINT "ファイル番号", "ハンドル", "モード"
PRINT "#1", FILEATTR(1, 2), FILEATTR(1, 1)
PRINT "#2", FILEATTR(2, 2), FILEATTR(2, 1)
```

```
ファイル番号    ハンドル      モード
#1                5             2
#2                6             1
```

FILES　（ファイル一覧の表示）

書式

FILES　[path]
　　　　↑ファイル名またはパス名

機能

path で指定したファイルまたはサブディレクトリ内のファイルの一覧を表示します．

path を省略すると，カレントディレクトリ内のすべてのファイル一覧が表示されます．

path にワールドカード(＊と？)を使うこともできます．

例

FILES ″B:＊.BAS ″
…ドライブBのファイルタイプ.BASを持つすべてのファイルを表示します．

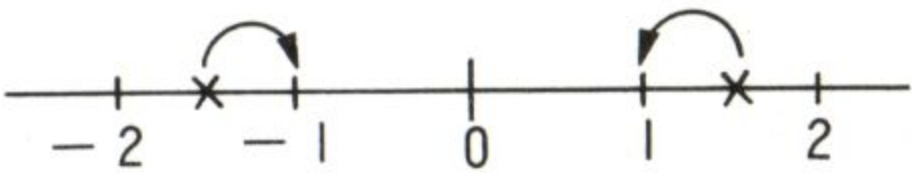
FILES ″¥BIN¥WORK ″
…サブディレクトリ¥BIN¥WORKのファイルを表示します．

FIX　（小数部切り捨て）　関数

書式

FIX(x)

機能

実数値 x の小数部を切り捨て，整数部だけを返します．

例

INT 参照．

FOR ~NEXT　（所定回反復）

書式

```
FOR   var = start TO end [STEP n]
       ↑      ↑        ↑          ↑
       |      |        |          └きざみ値
       |      |        └最終値
       |      └初期値
       └ループ変数
       ↓
NEXT  [var, …]
```

機能

FOR と NEXT で囲まれた範囲を所定の回数だけくり返します．所定の回数とは，初期値から最終値までループを1回くり返すたびに，きざみ値で示された値ずつ増加または減少させていった回数です．

初期値，最終値，きざみ値には，変数，定数，式を書くことができ，その値は正でも，負でも小数であってもかまいません．

きざみ値nが1のときだけSTEP nを省略できます．

FORのあとのループ変数と，NEXTのあとのループ変数とが異なってはいけません．ただし，NEXTのあとのループ変数は省略することができます．

FORループ内でループ変数の内容を変更したり，GOTO文でFORループ内に入ってはいけません．

FORループから強制的に抜けるにはEXIT FORを用います．

例

```
FOR J=1 TO 10
  :
NEXT J
```

……Jを1から始め1きざみで10になるまで，FORとNEXTで囲まれた部分をくり返す．

```
FOR A=5 TO 0 STEP -0.5
  :
NEXT A
```

……Aを5から始め，−0.5きざみで0になるまでくり返す．Aの値は5，4.5，4…1，0.5，0と変化する．

```
FOR J=1 TO 5
  FOR K=1 TO 10
    :
  NEXT K
NEXT J
```

……2重ループ．外側のループが1回まわるごとに内側のKループは10回まわる．したがって，5×10回くり返しが行われる．

FRE　（使用可能なメモリ・サイズの取得）　関数

書式

```
FRE(flag)
    └調べるメモリの種類
```

機　能

flagで示されるメモリの使用可能なサイズを取得します.

flag	機　能
−1	これから設定できる最大の非文字列配列のサイズをバイト数で返す
−2	スタックスペースの残りバイト数を返す
その他の数値	使用可能な文字列記憶領域のバイト数を返す
文字列	使用可能な文字列記憶領域のバイト数を返す

FREは使用可能なメモリ・サイズを取得する前に，使用可能な領域を圧縮処理して1つのブロックにします.

例

```
DECLARE SUB abc ()
PRINT TAB(14); "文字列配列", "非文字列配列", "スタック"
PRINT "main 設定前: ", FRE(0), FRE(-1), FRE(-2)

'$DYNAMIC
DIM a(100), a$(100)      ' ダイナミック配列の設定

PRINT "main 設定後: ", FRE(0), FRE(-1), FRE(-2)

abc

REM $STATIC
SUB abc
    DIM a, b, c      ' ローカル変数
    PRINT "procedure  : ", FRE(0), FRE(-1), FRE(-2)
END SUB
```

```
              文字列配列    非文字列配列    スタック
main 設定前:    46670        173822         1152
main 設定後:    46258        172994         1152
procedure  :    46258        172994         1130
```

FREEFILE　（未使用ファイル番号の取得）　関数

書　式

FREEFILE

機　能

次に使用できるファイル番号を取得します．この関数を使用すれば，プロシージャにおいて，すでに使用しているファイル番号を二重に指定してしまうことはありません.

例

```
' ---/* 作業用ファイルへのコピー */---
DECLARE SUB TempFile (n!)

OPEN "abc.dat" FOR INPUT AS #1

TempFile (1)

CLOSE #1

SUB TempFile (n)
    FileNum = FREEFILE ←―――――――次に使用できるファイル番号の取得
    OPEN "work.tmp" FOR OUTPUT AS FileNum
    WHILE NOT EOF(n)
        a$ = INPUT$(1, n) ←――――1文字入力
        PRINT #FileNum, a$; ←――――1文字出力
    WEND
    CLOSE FileNum
END SUB
```

…プロシージャTempFileは，work.tmpというファイルをファイル番号FileNumでオープンし，ファイル番号nのファイルを1文字単位でここにコピーする．

FUNCTION （関数プロシージャの定義）

書式

```
FUNCTION  name[( var [AS type], …)] [STATIC]
                 ↑        ↑仮引数 ↑仮引数の型
     ⋮           └プロシージャ名
   name = expression
            ↑関数の戻り値
END FUNCTION
```

機能

nameで示される関数プロシージャを定義します．関数の型は，nameのあとに型宣言文字(%, &, !, #, $)を付けるか， DEFtyp文を用いて名前のインプリシット宣言をして指定します．

(　)の中に仮引数を書きます．仮引数がなければ(　)を省略できます．

関数の戻り値は，プロシージャ名nameにexpressionで示す式の値を代入することで得られます．もしnameへの代入が置かれていなければ，数値型関数の戻り値は0，文字列型関数の戻り値はヌル文字列となります．

STATICを指定すると，関数プロシージャ内のローカル変数はすべて(SHARED宣言されている変数は除く) 静的変数となります．静的変数については第4章4-2の8を参照してください．

例

```
' ---/* データ入力関数 */---

DECLARE FUNCTION Dinput! (x!)

Sum = 0
WHILE Dinput(a) <> -1
    Sum = Sum + a
WEND
PRINT "合計="; Sum
END

FUNCTION Dinput (x)
    INPUT "data "; x
    IF x = -9999 THEN
        Dinput = -1       ' データの終わり
    ELSE
        Dinput = 1
    END IF
END FUNCTION
```

……関数プロシージャ Dinput は引数 x に入力データを返し，データ入力の終わりには関数戻り値 −1 を返すものとする．

```
data ? 1
data ? 2
data ? 3
data ? 4
data ? 5
data ? -9999
合計= 15
```

GET　（ランダム・ファイルからのリード）

書式

① GET　[#]n [,rec]

n：ファイル番号，rec：レコード番号

② GET　[#]n ,[rec][,var]

var：リード・データを格納する変数

機能

①ファイル番号 n のファイルから，FIELD 文で設定されているリード/ライト・バッファにデータをリードします．

②ファイル番号 n のファイルから，変数 var にデータをリードします．var を指定した場合は，FIELD 文でリード/ライト・バッファを設定しないでください．

rec はランダム・ファイルでは読み出すレコード番号，バイナリ・ファイルでは読み出しを始めるバイト位置です．レコード番号，バイト位置の先頭番号は 1 です．

rec を省略すると，ファイル現在位置（最後に GET/PUT でアクセスしたレコードの次のデータ，または SEEK で指定したデータ位置）のデータをリードします．rec は $1 \sim 2^{31}-1$ (2147483647) の範囲の値を指定します．

例

```
OPEN  "b: name.dat " FOR RANDOM AS #1 LEN =18
FIELD  #1, 16 AS Name$, 2 AS Year$
     ⋮
INPUT Rec
GET  #1, Rec  ←レコード番号 Rec のデータを FIELD で設定したバッファにリード

TYPE  Man
  Namae AS  STRING ＊16
  Year    AS  INTEGER
END  TYPE
DIM  A AS Man
OPEN  "b: name.dat " FOR RANDOM AS #1 LEN = LEN(A)
   ⋮
INPUT Rec
GET  #1, Rec,A  ←レコード番号 Rec のデータをレコード型変数 A にリード
```

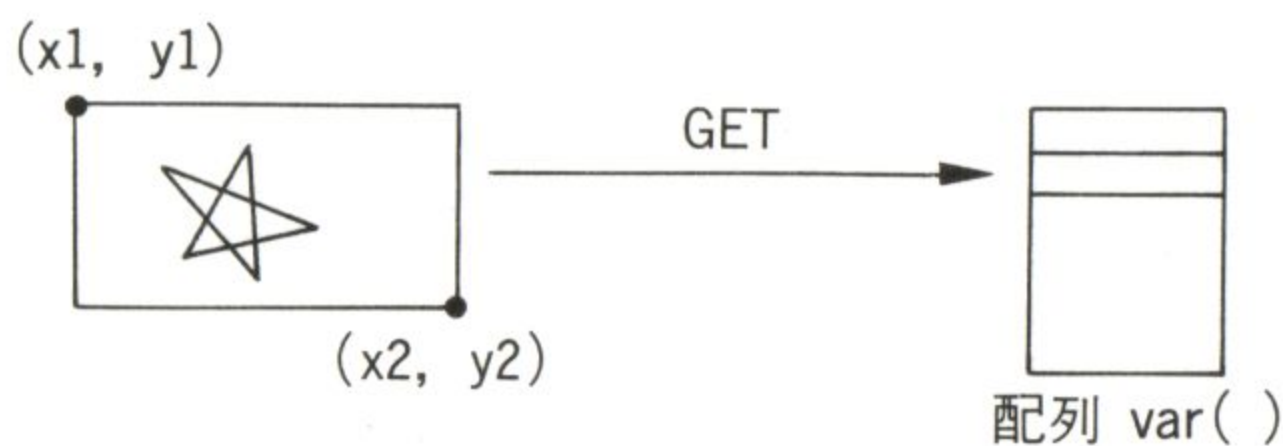
GET （画面イメージの保存）

書式

```
GET [STEP](x1,y1) - [STEP](x2,y2), var[(n)]
```

- (x1,y1)：左上座標
- (x2,y2)：右下座標
- var：配列名
- (n)：配列要素番号

機能

(x1,y1)－(x2,y2) で示される四角形のグラフィック画面のデータを配列 var に保存します．

(x1,y1)－(x2,y2) の画面データを保存するために必要なバイト数 B は，

```
B =4＋4＊ INT(((x2-x1＋1)＋7)/8)＊(y2-y1＋1)
```

で求められます．したがって必要な配列のサイズ A は，配列のデータ型によって次のように求められます．

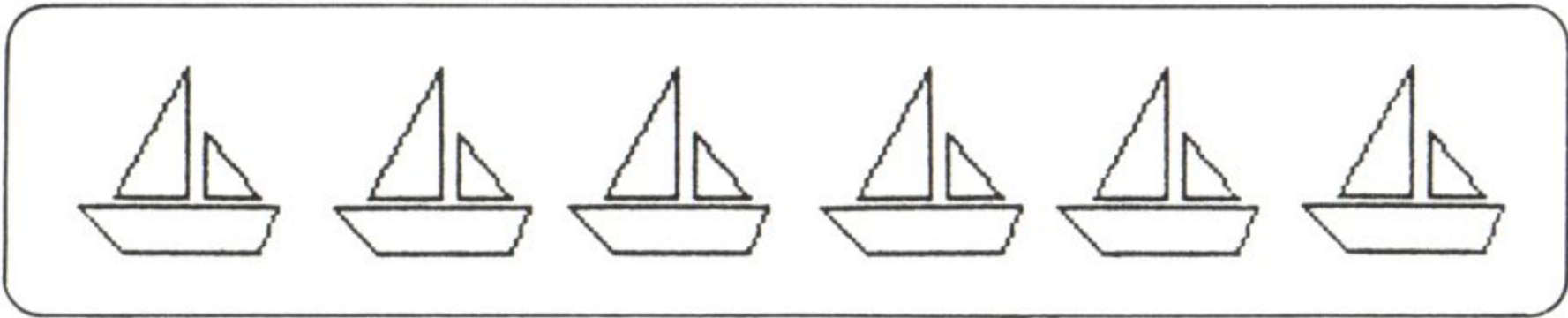
A = INT(B/M) + 1

- 2 : 整数型
- 4 : 倍長整数型
- 4 : 単精度実数型
- 8 : 倍精度実数型

STEP は相対座標を意味し， STEP (x1,y1) は現在位置からの相対距離，STEP (x2,y2) は (x1,y1) からの相対距離となります．

n は，イメージを保存する配列の格納開始要素を示す値です．省略すると配列先頭からになります．

例

```
' ---/* 画面イメージのＧＥＴ／ＰＵＴ */---

SCREEN 0: CLS
x = 45: y = 41  ←―――――――――――― 領域の横・縦のドット数
byte = 4 + 4 * ((x + 7) ¥ 8) * y     ' 必要なバイト数
s = byte ¥ 2 + 1                     ' 必要な配列要素の数
DIM p%(s)                            ' イメージ用配列

COLOR 1                                                     ┐
LINE (0, 30)-(10, 40): LINE -(40, 40)                       │
LINE -(44, 30): LINE -(0, 30)                               │←―ヨットの描画
LINE (24, 0)-(8, 28): LINE -(24, 28): LINE -(24, 0)         │
LINE (28, 14)-(28, 28): LINE -(40, 28): LINE -(28, 14)      ┘

GET (0, 0)-(44, 40), p%  ←――――― ヨット・パターンの取得

CLS
FOR x = 0 TO 300 STEP 60  ┐
    PUT (x, 0), p%, PSET  ┘←――― ヨットを横に表示していく
NEXT x
```

GOSUB　（サブルーチンの呼び出し）

書式

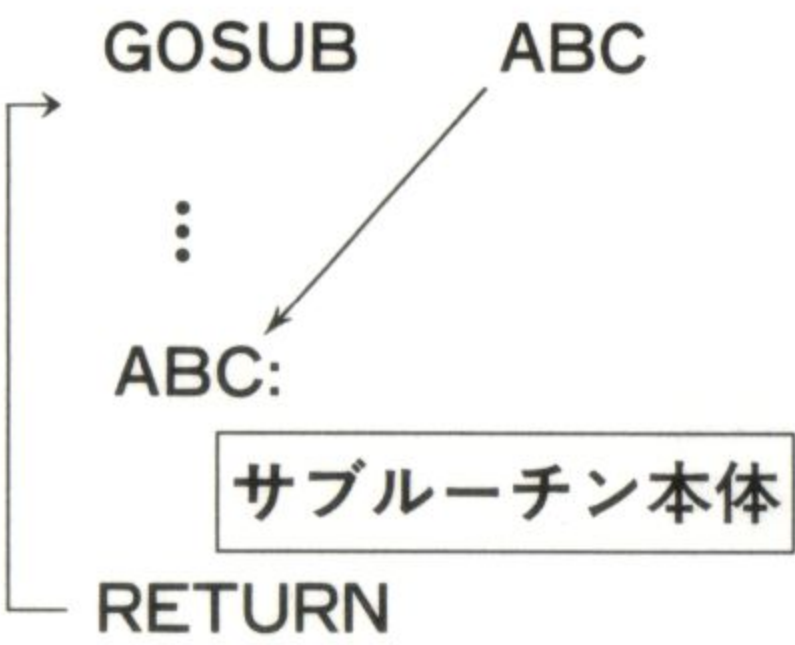
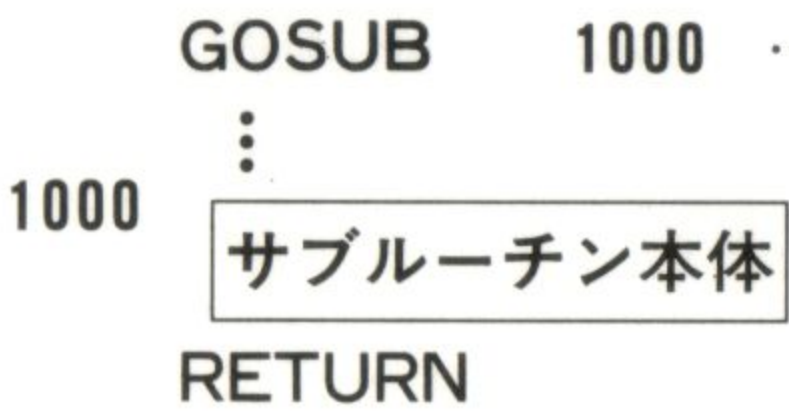
GOSUB　line

　　　　└─ 飛び先

機能

line で示されるサブルーチンを呼び出します．line にはラベルまたは行番号を指定します．

サブルーチンからは RETURN で戻ります．

例

```
┌→ GOSUB  ABC   …ラベル ABC で示されるサブルーチンへ
│    ⋮           分岐する．サブルーチンの RETURNに
│                より，GOSUB の次の文に戻る．
│  ABC:
│    サブルーチン本体
└─ RETURN
```

```
     GOSUB  1000  …1000行のサブルーチンへ分岐する．
       ⋮
1000 サブルーチン本体
     RETURN
```

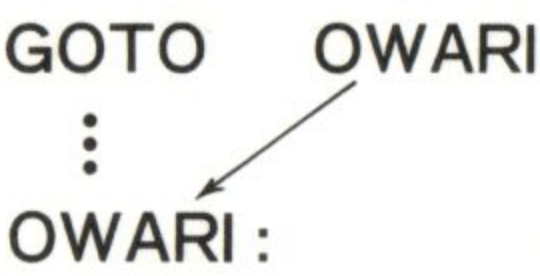
GOTO （分岐）

書式

```
GOTO  line
      ↑─飛び先
```

機能

line で示される位置へ分岐します．line にはラベルまたは行番号を指定します．

例

```
GOTO  OWARI   …ラベル OWRI で示される位置
  ⋮             へ分岐する．
OWARI :
```

HEX$ （数値を16進文字列に変換） 関数

書式

```
HEX$(n)
     ↑─数値
```

機能

数値 n を16進文字列に変換して返します．

例

```
A$ = HEX$(255)  …FF が返される．
```

IF THEN ELSE　（条件判断）

書式

```
① IF  p     THEN  s1  [ELSE s2]
      ↑           ↑          ↑─文
      └─条件式    └─文

      ┌─条件式
      ↓
② IF  p1  THEN
       s1  ←文
  [ELSEIF  p2  THEN
       s2 ]
       ⋮
  [ELSE
       sn ]
  END IF
```

機能

①単純 IF．条件 p を満たせば s1を，満たさなければ s2を実行します．s1,s2にはコロン(：)で区切ってマルチ・ステートメントを書くことができます．単純 IF 文は1行に収まる範囲です．

②ブロック IF．条件 p1を満たせばブロック s1を，そうでなく p2を満たせばブロック s2を，…，いずれも満たさなければブロック sn を実行します．
ELSEIF 節，ELSE 節は省略できますが，END IF は省略できません．
ブロック s1～sn は複数行からなる文の集まりです．

例

```
IF  A>0   THEN  SP=SP+A   ELSE  SM=SM+A
```

…A>0 なら SP=SP+A を，そうでなければ SM=SM+A を行う．

```
IF  A>0   THEN
     SP=SP+A        ←Ⓐ
     PRINT SP
ELSE
     SM=SM+A        ←Ⓑ
     PRINT SM
END IF
```

…A>0ならブロックⒶを，そうでないならブロックⒷを実行する．

```
IF    A>80    THEN
      ブロックA
ELSEIF   A>70   THEN
      ブロックB
ELSEIF   A>60   THEN
      ブロックC
ELSE
      ブロックD
END IF
```

…A＞80ならブロックAを，
そうでなくA＞70ならブロックBを，
そうでなくA＞60ならブロックCを，
それら以外ならブロックDを実行する．

INKEY$ （待ったなしのキー入力） 関数

書式

INKEY$

機能

入力されたキー文字を返します．キー入力がされていなければ，ヌル文字列を返します．

キーから読んだコードが漢字コードまたは拡張コードの場合，2バイト文字を返します．

INKEY$は入力文字を画面上にエコーバックしません．

次のキーはINKEY$では読み込むことはできません．

CTRL ＋ C	**プログラムの終了**
COPY	**画面のコピー**

ただし，コンパイル時にD/デバッグコードまたは/dオプションを指定していない独立型の.EXEプログラムは，CTRL＋Cを読むことができます．

例

```
CLS
DO
    a$ = INKEY$
    IF a$ <> "" THEN PRINT ">";
LOOP WHILE a$ <> CHR$(27)
```

…キーを押すたびに“＞”を表示する．ESCでループから抜ける．

INP （ポートから1バイト入力） 関数

書式

INP(n)
　　└ポート番号

機能

n番(0～65535)のポートから1バイトのデータを入力します.

例

Lputcは，プリンタポートに直接データを出力する関数です.
Lputsは，Lputcを用いて文字列をプリンタに出力します.

```
DECLARE SUB Lputc (c%)
DECLARE SUB Lputs (a$)

Lputs ("Hello Quick")
Lputc (&HD): Lputc (&HA)

SUB Lputc (c%)        ' 直接プリンタ出力
    WHILE (INP(&H42) AND 4) <> 4        ┐←第2ビットがセット(1)
    WEND                                ┘ されるまで待つ
    OUT &H40, c%
    OUT &H46, 14
    OUT &H46, 15
END SUB

SUB Lputs (a$)        ' 直接文字列出力
    FOR k = 1 TO LEN(a$)
        c% = ASC(MID$(a$, k, 1))
        Lputc c%
    NEXT k
END SUB
```

INPUT$ （指定したバイト数分の文字列のリード） 関数

書式

INPUT$(size[, #n])
　　　　↑　　　└ファイル番号
　　　　└リードバイト数

機能

ファイル番号nのファイルからsizeバイトの文字をリードします.
ファイル番号nを省略すると，標準入力(通常キーボード)からリードします.

例

次のプログラムは，ファイルから1バイト単位でリードし，スクリーンに表示していくものです.
MS-DOSのテキスト・ファイルの行末はCR・LF(&H0D,&H0A)の2バイトで構成され，QBでこの文字をスクリーンに2回表示すると改行が2度行われるため，最初の&H0Dは出力しないようにしました.

```
' ---/* ファイルのタイプ */---

CLS
INPUT "FileName "; File$
OPEN File$ FOR INPUT AS #1

DO WHILE NOT EOF(1)
    a$ = INPUT$(1, #1) ←──────── 1文字(バイト)リード
    IF a$ <> CHR$(&HA) THEN PRINT a$;
LOOP
CLOSE #1
```

INPUT¥　（指定した長さの文字のリード）　関数

書式

INPUT¥(size[, #n])

- size ← リードする文字数
- n ← ファイル番号

機能

ファイル番号nのファイルから，sizeで示される個数の文字をリードします．1バイト文字なら1バイトを，2バイト文字なら2バイトを1文字として扱います．INPUT$はバイト単位のリードです．

例

次のプログラムは，ファイルから1文字単位でリードし，スクリーンに表示して行くものです．INPUT$で示した例と同じものですが，こちらは，2バイト文字を含むファイルに対しても正常に動作します．

```
' ---/* ファイルのタイプ */---

CLS
INPUT "FileName "; File$
OPEN File$ FOR INPUT AS #1

DO WHILE NOT EOF(1)
    a$ = INPUT¥(1, #1) ←──────── 1文字リード
    IF a$ <> CHR$(&HA) THEN PRINT a$;
LOOP
CLOSE #1
```

INPUT　（キーボードからのデータ入力）

書式

INPUT [;] ["msg" {; | , }] var,…

- "msg" ← メッセージ文字列
- var ← 変数

機　能

変数 var にキーボードからデータを入力します.

INPUT 直後にセミコロン(;)を置くと，データ入力時に[↵]を押してもカーソルは同じ行にとどまります.

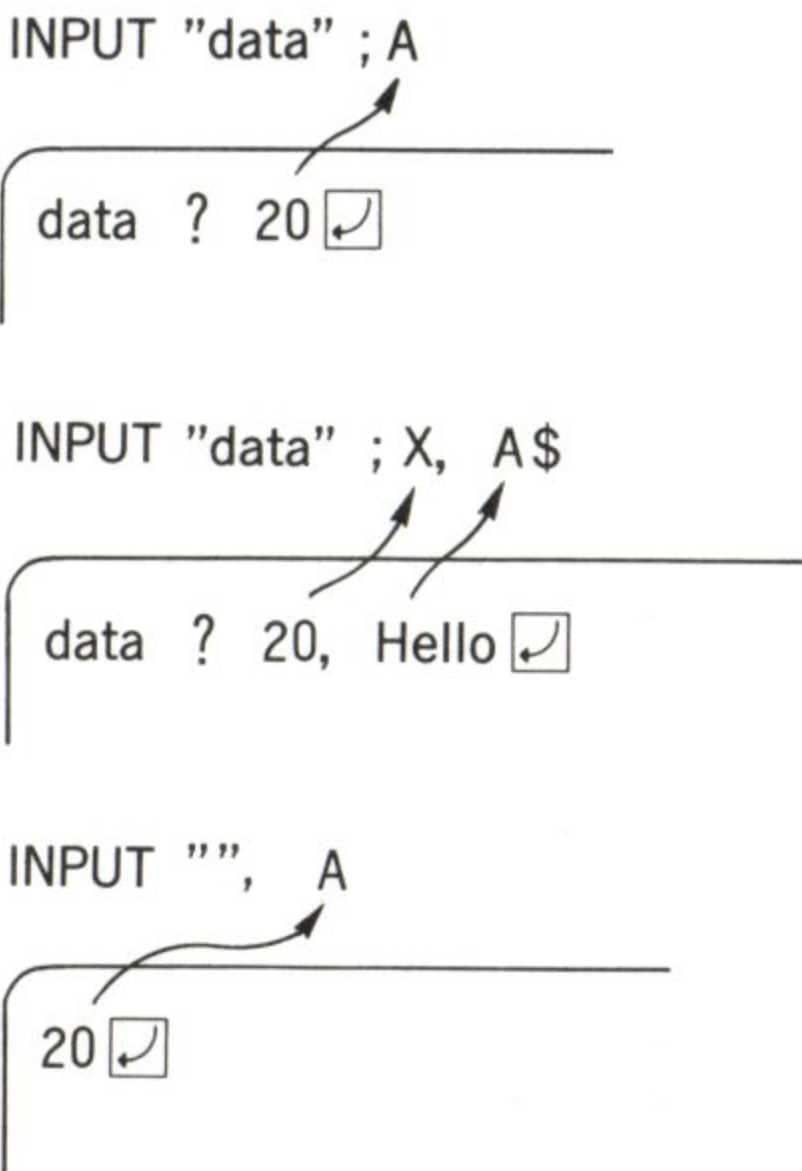
msg はメッセージ文字列で，その後に置く“;”と“,”は機能が異なります.

;　… msg のあとに?を表示する.
,　… msg のあとに?を表示しない.

例

```
INPUT "data" ; A

data ? 20↵
```

```
INPUT "data" ; X, A$

data ? 20, Hello↵
```

```
INPUT "", A

20↵
```

INPUT #（シーケンシャル・ファイルからのリード）

書　式

```
INPUT #n, var,…
```

n … ファイル番号
var … 変数

機　能

ファイル番号 n のシーケンシャル・ファイルから，データを変数 var にリードします.

例

```
' ---/* シーケンシャル・ファイルの読み込み */---

OPEN "man.dat" FOR INPUT AS #1

PRINT "Name", "Year"
DO WHILE NOT EOF(1)
    INPUT #1, Name$, Year
    PRINT Name$, Year
LOOP
CLOSE #1
```

INSTR　　（文字列の検索）　関数

書式

INSTR ([start,]str, key)

- start — 検索開始位置
- str — 被検索文字列
- key — 見つける文字列

機能

文字列 str 中から key で示される文字列を探し，その位置(バイト位置)を返します．start は検索開始位置で，省略すると先頭から検索を開始します．

検索文字列が見つからなければ 0 を返します．

例

INSTR("Quick␣BASIC", "BAS")… 7 を返します．
次のプログラムは文字列中の空白の位置を表示するものです．

```
Text$ = "THis is a pen"
p = INSTR(Text$, " ")
DO WHILE p <>0
    PRINT p
    p = INSTR(p + 1, Text$, " ")
LOOP
```

（p + 1：空白の次の位置を開始点にする）

```
5
8
10
```

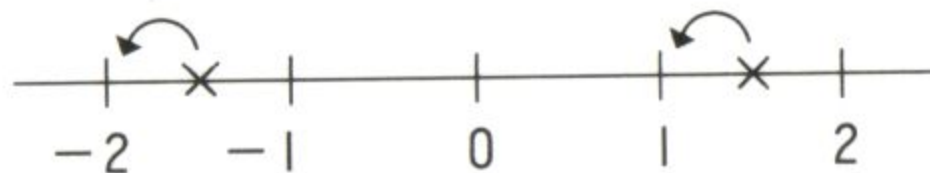
INT　　（小数部切り捨て）　関数

書式

INT(x)

機能

実数値 x の小数部を切り捨て，それよりも小さい整数を返します．

−2　−1　0　1　2

例　次の例はINT，CINT，FIXの違いを比較するものです．

```
PRINT "x", "INT(x)", "CINT(x)", "FIX(x)"
FOR k = 1 TO 4
    READ x
    PRINT x, INT(x), CINT(x), FIX(x)
NEXT k
 DATA -2.6,-2.3,2.6,2.3
```

```
x              INT(x)      CINT(x)     FIX(x)
-2.6           -3          -3          -2
-2.3           -3          -2          -2
 2.6            2           3           2
 2.3            2           2           2
```

IOCTL$（デバイス・ドライバから制御文字列を取得）　関数

書式

```
IOCTL$([#]n)
       ↑_デバイス・ドライバのファイル番号
```

機能　オープンされているデバイス・ドライバから制御文字列を取得します．

IOCTL　（制御文字列をデバイス・ドライバに送る）

書式

```
IOCTL [#]n, str
       ↑    ↑_制御文字列
       └デバイス・ドライバのファイル番号
```

機能　オープンされているデバイス・ドライバに制御文字列を送ります．

JIS$　（文字をJISコードに変換）　関数

書式

```
JIS$(str)
     ↑_文字列
```

機能　文字列strの先頭文字のJISコードを返します．先頭文字が1バイト文字ならASCIIコードを返します．

例

```
a$ = "ABC日本語"
FOR k = 1 TO KLEN(a$)
    b$ = KMID$(a$, k, 1)
    PRINT b$; "="; JIS$(b$)
NEXT k
```

```
A=65
B=66
C=67
日=467C
本=4B5C
語=386C
```

KEXT$ （1バイト文字または2バイト文字の抽出） 関数

書式

KEXT$ (str, func)
- str ── 文字列
- func ── 抽出する文字の種類

機能

func =0のときは，str から1バイト文字だけを抜き出して返し，func =1のときは，str から2バイト文字だけを抜き出して返します．指定した文字列が存在しなければ，ヌル文字列が返されます．

例

```
a$ = "日本語ABC入門123"
PRINT KEXT$(a$, 0), KEXT$(a$, 1)
```

```
ABC123        日本語入門
```

KEY （ファンクション・キー制御）

書式

① KEY n, str
- n ── ファンクション・キーの番号（1～10）
- str ── ファンクション・キーに割り当てる文字列

② KEY LIST
③ KEY ON
④ KEY OFF

機　能

①n(1～10)で示されるファンクション・キーに文字列を割り当てます．文字列は最大15文字までで，それを越えた文字は捨てられます．
②ファンクション・キーに割り当てられている文字列の一覧を表示します．
③画面の最下行に各ファンクション・キーに割り当てられている最初の6文字を表示します．
④画面の最下行へのファンクション・キーの表示を行いません．

KEY により設定したファンクション・キーが機能するのは，プログラムの実行中で，QB 環境下ではありません．

例

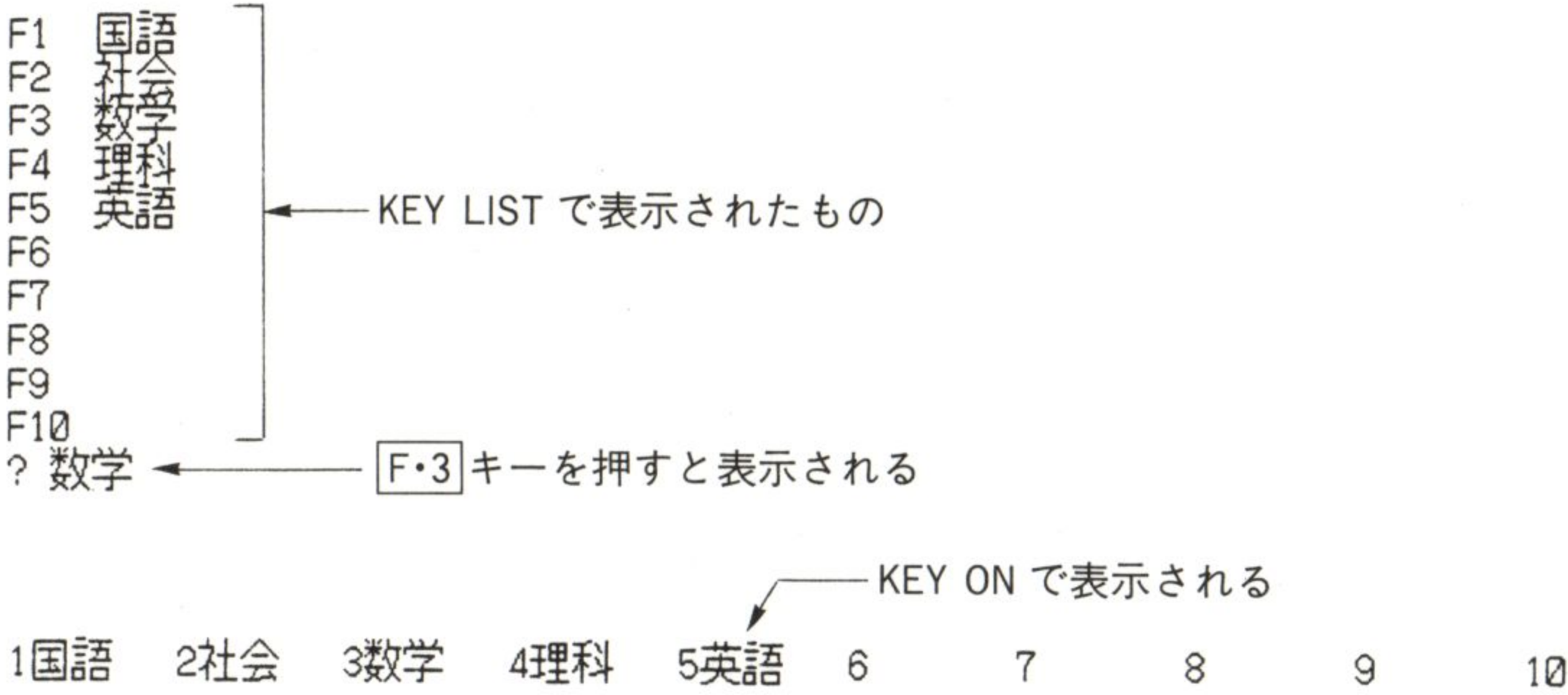
```
CLS
FOR k = 1 TO 5
    READ a$
    KEY k, a$
NEXT K
DATA 国語,社会,数学,理科,英語

KEY ON
KEY LIST

INPUT Text$
```

← F･1 ～ F･5 にデータを割り当てる

```
F1  国語
F2  社会
F3  数学
F4  理科
F5  英語
F6
F7
F8
F9
F10
? 数学

1国語   2社会   3数学   4理科   5英語   6      7      8      9      10
```

← KEY LIST で表示されたもの

← F･3 キーを押すと表示される

← KEY ON で表示される

KEY(n) （キー・トラッピングの開始／禁止／停止）

書　式

① KEY(n)　ON
② KEY(n)　OFF
③ KEY(n)　STOP
　　　　└キー番号

機　能

①指定したキーのトラッピングを開始
②指定したキーのトラッピングを禁止
③指定したキーのトラッピングを停止．停止中に発生したトラッピングは記憶されており，トラッピングが許可(開始)されるとただちにイベント処理ルーチンに分岐します．

キー番号 n とキーの関係は次のとおりです．

n	キー
1〜10	[F・1]〜[F・10]
11	[↑]
12	[←]
13	[→]
14	[↓]
15〜25	ユーザ定義キー

ユーザ定義キーは KEY 文を用いて次のように定義します．

KEY n，CHR$(keyflag)＋CHR$(scancode)

keyflag の値は以下のとおりです．

keyflag	キー
&H00	キーボードフラグなし
&H01	[SHIFT]
&H02	[CAPS]
&H04	[カナ]
&H08	[GRPH]
&H10	[CTRL]

scancode は次ページの表のとおりです．

たとえば，

KEY 15，CHR$(&H10)＋CHR$(&H1E)

は，[CTRL]＋[S]を15番のキーに割り当て，

KEY 15，CHR$(&H00)＋CHR$(&H1E)

は，[S]を15番のキーに割り当てます．

● scancode

キー	コード	キー	コード	キー	コード	キー	コード
ESC	00	P	19	/ ?	32	2	4B
1 !	01	@ ~	1A	—	33	3	4C
2 "	02	[{	1B	スペース	34	=	4D
3 #	03	改行	1C	XFER	35	0	4E
4 $	04	A	1D	ROLL UP	36	,	4F
5 %	05	S	1E	ROLL DOWN	37	.	50
6 &	06	D	1F	INS	38	NFER	51
7 '	07	F	20	DEL	39	STOP	60
8 (	08	G	21	↑	3A	COPY	61
9)	09	H	22	←	3B	f・1	62
0	0A	J	23	→	3C	f・2	63
- =	0B	K	24	↓	3D	f・3	64
^ ‘	0C	L	25	HOMECLR	3E	f・4	65
¥ |	0D	; +	26	HELP	3F	f・5	66
BS	0E	: *	27	—	40	f・6	67
TAB	0F	] }	28	/	41	f・7	68
Q	10	Z	29	7	42	f・8	69
W	11	X	2A	8	43	f・9	6A
E	12	C	2B	9	44	f・10	6B
R	13	V	2C	*	45	SHIFT	70
T	14	B	2D	4	46	CAPS	71
Y	15	N	2E	5	47	カナ	72
U	16	M	2F	6	48	GRPH	73
I	17	, <	30	+	49	CTRL	74
O	18	. >	31	1	4A		

例

```
KEY 15, CHR$(&H10) + CHR$(&H1E)  'ユーザ定義キー   ←— CTRL + S をキー番号15に定義

KEY(1)ON                 'トラップの開始
KEY(15)ON
ON KEY(1) GOSUB Key1     'トラップ・ルーチンの設定
ON KEY(15) GOSUB Key2

n = 1: m = 1
WHILE 1          '無限ループ
WEND

Key1:
    PRINT m; ":now trap key1"
    m = m + 1
RETURN

Key15:
    PRINT n; ":now trap key15"
    n = n + 1
RETURN
```

```
1 :now trap key1
1 :now trap key15
2 :now trap key1
3 :now trap key1
```

KILL　　（ファイルの削除）

書式

KILL filename
　　　　↑ファイル名

機能

filenameで示されるファイルを削除します．ファイル名にワイルドカード(＊，？)を使うこともできます．

例

KILL "B: ＊.BAS"
…ドライブBのファイルタイプ.BASのファイルをすべて削除する．

KINSTR　（文字列の検索：INSTRの漢字対応版）　関数

書式

KINSTR([start,] str, key)

- key：見つける文字列
- str：被検査文字列
- start：検索開始位置

機能

文字列str中からkeyで示される文字列を探し，その位置(1バイト文字，2バイト文字とも1文字として数える)を返します．startは検索開始位置で，省略すると先頭から検索を開始します．

検索文字が見つからなければ0を返します．

例

```
' ---/* 文字列の検索 */---

Key$ = "日記"
FOR k = 1 TO 4
    READ Book$
    IF KINSTR(Book$, Key$) <> 0 THEN PRINT Book$
NEXT k
DATA 土佐日記,源氏物語,十六夜日記,方丈記
```

…Book$にリードしたデータの中に"日記"という文字が入っているものだけを表示する．

```
土佐日記
十六夜日記
```

KLEN　（文字列長さの取得：LEN の漢字対応版）　関数

書式

```
KLEN(str)
     └─文字列
```

機能

文字列の長さを文字数で返します．1バイト文字，2バイト文字とも1文字として数えます．

例

```
a$ = "日本語ABC"

n = LEN(a$)
m = KLEN(a$)

PRINT "バイト数        ="; n
PRINT "文字数          ="; m
PRINT "2バイト文字の数="; n - m
```

```
バイト数        = 9
文字数          = 6
2バイト文字の数 = 3
```

KMID$　（中間文字列の取得：MID$の漢字対応版）　関数

書式

```
KMID$(str, start[, n])
                    └─取り出す文字数
            └─取り出す開始位置
      └─文字列
```

機能

文字列 str の start 位置から，n 個の文字を取り出して返します．1バイト文字，2バイト文字とも1文字として数えます．

n を省略すると，文字列の終わりまで取り出します．

例

```
a$ = "日本語abc"

FOR k = 1 TO KLEN(a$)
    PRINT KMID$(a$, k, 1)
NEXT k
```

```
日
本
語
a
b
c
```

KMID$　（文字列の置き換え：MID$の漢字対応版）

書式

```
KMID$(strvar, start[, n])= strexpression
                                 └置き換える文字列
                      └置き換える文字数
               └置き換え開始位置
     └置き換えられる文字列
```

機能

文字列変数 strvar の start 位置以後の文字を，strexpression で置き換えます．n を指定すると，strexpression の先頭から n 文字分を置き換えます．

例

```
Text$ = "これは船      "
Key$ = "船"
Rep$ = "電車"
KMID$(Text$, KINSTR(Text$, Key$)) = Rep$
PRINT Text$
```

…Text$中から Key$の文字列を探し，Rep$で置き換える．船の代わりに電車が入るのではなく，船␣の代わりに電車が入る．

```
これは電車
```

KPOS　（指定文字位置までのバイト数の取得）　関数

書式

```
KPOS(str, n)
           └文字位置
      └文字列
```

機能

文字列 str の n 番目の文字(1バイト文字，2バイト文字を1文字と数える)が，先頭から何バイト目になるかを求めて返します．

例

```
KPOS("日本語 abc", 5)   … 8を返す．
                └ 5番目の文字
```

KTN$　（文字を句点コードまたは ASCII コードに変換）　関数

書式

```
KTN$ (str)
      └文字列
```

機能

文字列 str の先頭1文字が2バイト文字のときは，それに対応する句点コードを，1バイト文字のときは，それに対応する ASCII コードを文字列の形で返します．

例

```
a$ = "０Ａア亜日本語"

PRINT TAB(14); "句点コード", "ＪＩＳコード"
FOR k = 1 TO KLEN(a$)
    b$ = KMID$(a$, k, 1)
    PRINT b$, KTN$(b$), JIS$(b$)
NEXT k
```

	句点コード	ＪＩＳコード
０	0316	2330
Ａ	0333	2341
ア	0502	2522
亜	1601	3021
日	3892	467C
本	4360	4B5C
語	2476	386C

LBOUND　（配列添字の下限値の取得）　関数

書式

LBOUND(array[, d])

- array：配列名
- d：添字の次元

機能

配列 array の d 次の添字の下限値を取得します.

LBOUND, UBOUND は，引数渡しされた，配列のサイズを取得するのに使います.

DIM　A(−10　TO　20, 5　TO　10)

LBOUND(A, 1)　…　−10 を返す.
LBOUND(A, 2)　…　5 を返す.

例

プロシージャ SetArray は2次元配列を受けとり，各要素を dat で初期化します.

```
DECLARE SUB SetArray (b!(), dat!)
DIM a(-2 TO 2, 3 TO 5)

SetArray a(), -1


SUB SetArray (b(), dat)
    FOR j = LBOUND(b, 1) TO UBOUND(b, 1) ←—— 第1次の添字の下限と上限
        FOR k = LBOUND(b, 2) TO UBOUND(b, 2) ←— 第2次の添字の下限と上限
            b(j, k) = dat
        NEXT k
    NEXT j
END SUB
```

LCASE$ （小文字に変換） 関数

書 式

```
LCASE$(str)
       ↑文字列
```

機 能

文字列中の大文字を小文字に変換して返します．対象となる文字は1バイト文字，2バイト文字のA～Zです．

例

```
INPUT A$
IF LCASE$(A$) = "yes" THEN～
```

…A$にYes，YES，などが入力されてもyesとして比較できる．

LEFT$ （左部分文字列の取得） 関数

書 式

```
LEFT$(str, n)
      ↑    ↑左端からの文字数
      └文字列
```

機 能

文字列strの左端からn文字(バイト)を取り出します．nが文字列の長さより大きければ文字列全体を返し,nが0ならヌル文字列を返します．

例

```
A$ = "Hello BASIC"
B$ = LEFT$(A$, 5) … Helloが返される.
```

LEN （文字列長さの取得） 関数

書 式

```
LEN(str)
    ↑文字列
```

機 能

文字列strの長さを返します．2バイト文字は2文字と数えます．日本語を含む文字列にはKLENを使用してください．

例

```
LEN("Hello")   … 5を返す.
LEN("日本語 ABC")… 9を返す.
```

LEN　（文字のサイズの取得）　関数

書式

```
LEN(var)
    └─変数名
```

機能

変数がメモリ上で使用するバイト数を取得します．

例

```
TYPE Man
      Name AS STRING * 16
      Year AS INTEGER
END TYPE
DIM A AS Man

LEN(A)  … 16+2=18バイトが返される．
```

LET　（代入）

書式

```
[LET] var = expression
       │          └─式
       └─変数
```

機能

変数に式を代入するための文です．一般に LET は省略し，

var = expression

だけで代入文として扱われます．

LINE　（直線の描画）

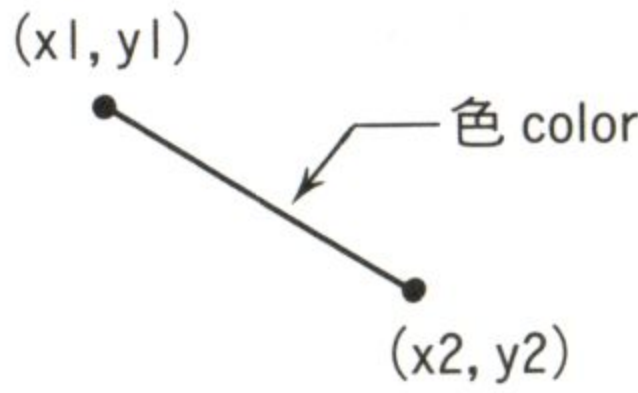
書式

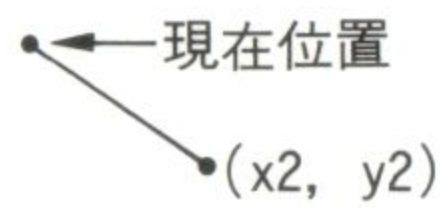
```
LINE [[STEP](x1, y1)] - [STEP](x2, y2), [color],[{B|BF}][,style]
            │                  │         │       │        └─ラインスタイル
            └─始点             └─終点    └─色    └─Box, Box Fill
```

機能

①通常の直線

LINE (x1, y1) - (x2, y2) [,color]

色 color を省略すると COLOR 文で設定されている色で描きます．

②現在位置からの描画

LINE -(x2, y2)[,color]

現在位置
(x2, y2)

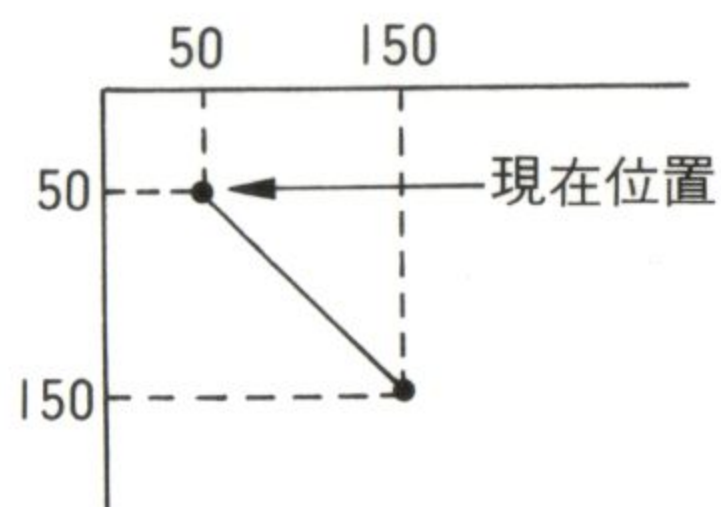
始点座標(x1, y1)を省略すると，現在位置からの描画となります．

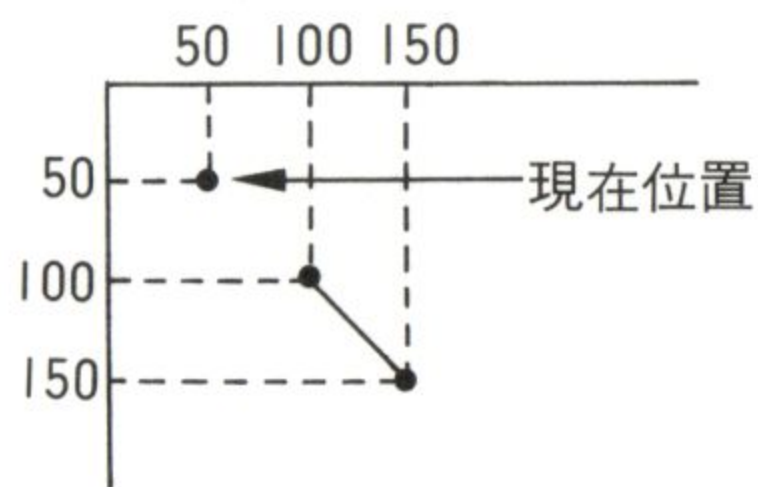
③相対指定

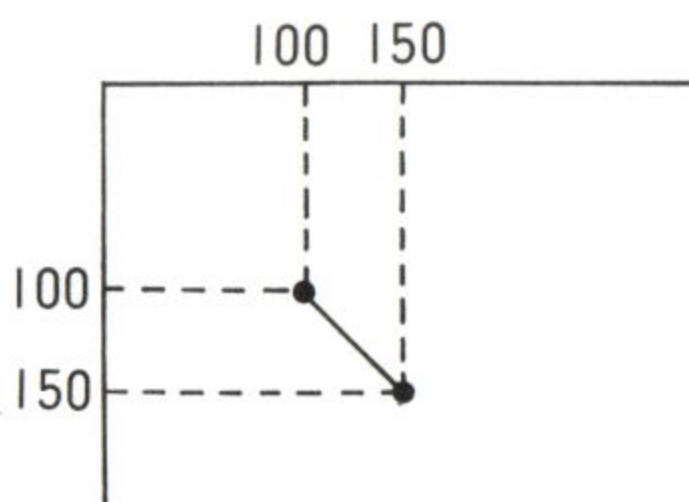
LINE [[STEP](x1, y1)]-[STEP](x2, y2)[,color]

第1座標にSTEPを付けると，現在位置からの距離，第2座標にSTEPを付けると，第1座標からの距離となります．

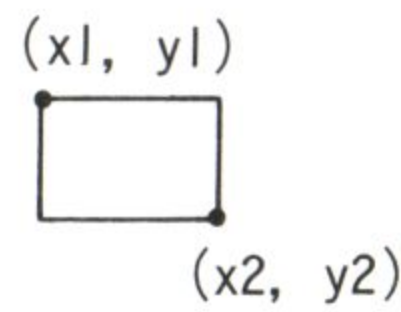
LINE -STEP(100, 100)

50 150
50 現在位置
150

LINE STEP(50, 50)-STEP(50, 50)

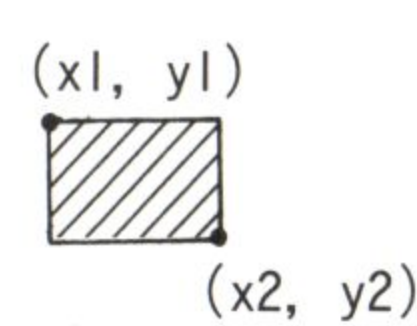
50 100 150
50 現在位置
100
150

LINE (100, 100)-STEP(50, 50)

100 150
100
150

④ B(Box), BF(Box Fill)

B指定すると(x1, y1)-(x2, y2)を対角とする四角形を描きます．座標にSTEPを指定することもできます．

LINE (x1, y1)-(x2, y2),[color], B

(x1, y1)
(x2, y2)

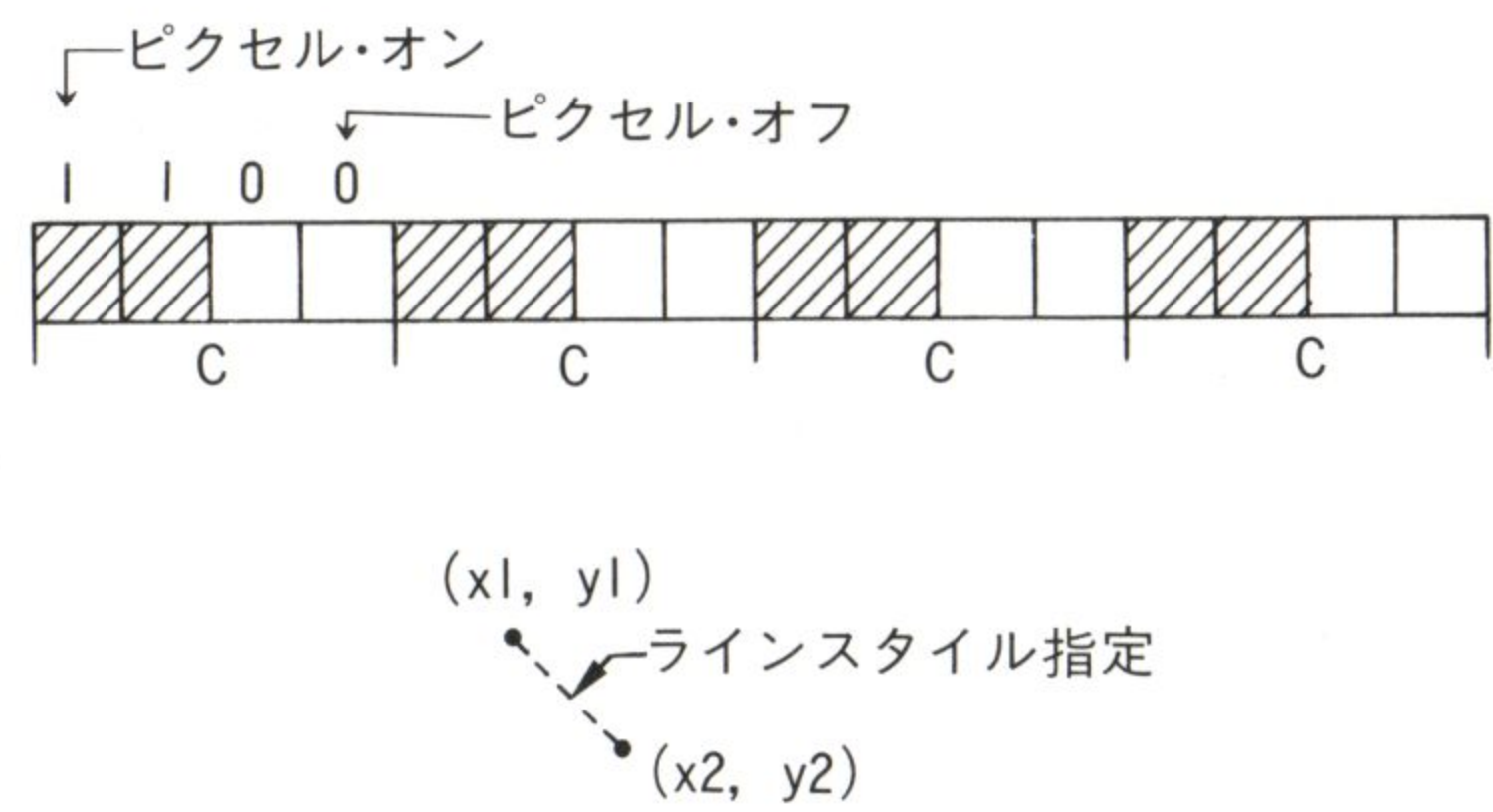
BF指定すると(x1, y1)-(x2, y2)を対角とする四角形を描き，その中をぬりつぶします．座標にSTEPを指定することもできます．

LINE (x1, y1)-(x2, y2),[color], BF

(x1, y1)
(x2, y2)

⑤ラインスタイルの指定

styleに次のような16ビットのビットパターン値を与えることで，線のスタイル(点線，破線など)を指定できます．

ピクセル・オン
ピクセル・オフ
1 1 0 0
C C C C

(x1, y1)
ラインスタイル指定
(x2, y2)

LINE (x1, y1) - (x2, y2),[color],, &HCCCC

ラインスタイルの指定はBFオプションに対しては機能しません．

例

```
' ---/* 直線の描画  */---

SCREEN 0: CLS

LINE (10, 10)-(200, 10), 1 ←――――――――― ①
LINE (10, 20)-(200, 20), 2, , &HCCCC ←―― ②
LINE (10, 30)-(200, 200), 3, B ←――――――― ③
LINE (30, 40)-(100, 110), 4, BF ←――――――― ④
LINE (110, 110)-(190, 190), 5, B, &HF00F ←― ⑤
```

①
②
③
④
⑤

LINE INPUT （キーボードから1行入力）

書　式

LINE INPUT [;]["msg";] strvar

メッセージ（msg）　文字列変数（strvar）

機　能

キーボードから入力された1行のデータをstrvarに格納します．データ中にカンマ，スペース，二重引用符などを含めることができます．[↲]のみ入力するとヌル文字列を返します．

msgは表示されるメッセージです．INPUT文と異なり，msg中に?を含ませないかぎり，?を表示しません．

INPUT直後に“；”を置くと，[↲]を押したあともカーソルは同じ行にとどまります．

例

キーボードから入力した1行をファイルに書き込みます．

入力の終わりは[CTRL]＋[Z][↲]とします．[CTRL]＋[Z]は&H1Aというコードです．

```
OPEN "text.dat" FOR OUTPUT AS #1

LINE INPUT a$
DO WHILE MID$(a$, 1, 1) <> CHR$(&H1A) ←―― 入力行の先頭文字の判定
    PRINT #1, a$
    LINE INPUT a$
LOOP
CLOSE #1
```

LINE INPUT＃（シーケンシャル・ファイルから1行単位で入力）

書式

LINE INPUT #n, strvar
（n：ファイル番号、strvar：文字列変数）

機能

ファイル番号nのシーケンシャル・ファイルのファイル現在位置から1行をリードし，strvarに格納します．1行の終わりはCR・LFコードで判定されます．

例

1行単位でファイルからリードし，行番号を付けて表示するものです．このリストはこのプログラム自身で出力したものです．

```
 1: ' ---/* 行番号付きファイル・タイプ */---
 2:
 3: INPUT "FileName"; f$
 4: OPEN f$ FOR INPUT AS #1
 5:
 6: n = 1
 7: LINE INPUT #1, a$   ←――― 1行リード
 8: DO WHILE NOT EOF(1)
 9:     PRINT USING "###: "; n;   ←――― 行番号の表示
10:     PRINT a$
11:     n = n + 1   ←――― 行番号を1つ進める
12:     LINE INPUT #1, a$
13: LOOP
14: CLOSE #1
```

LOC（ファイル現在位置の取得）　関数

書式

LOC(n)
（n：ファイル番号）

機能

ランダム・ファイルでは，最後にリード/ライトしたレコード番号を返します．

シーケンシャル・ファイルでは，現在位置のバイト数を128で割った値を返します．

バイナリ・ファイルでは，最後にリード/ライトしたバイト位置を返します．

COM(通信)デバイスでは入力バッファにある文字数を返します．

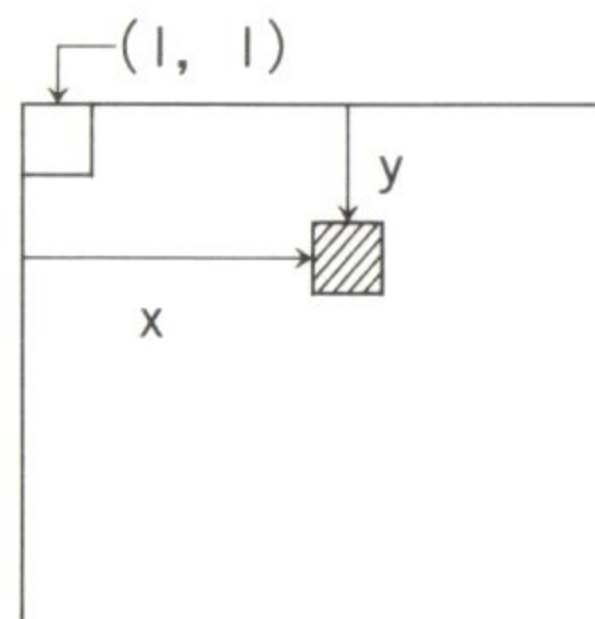
LOCATE　　　（カーソルを指定位置に移す）

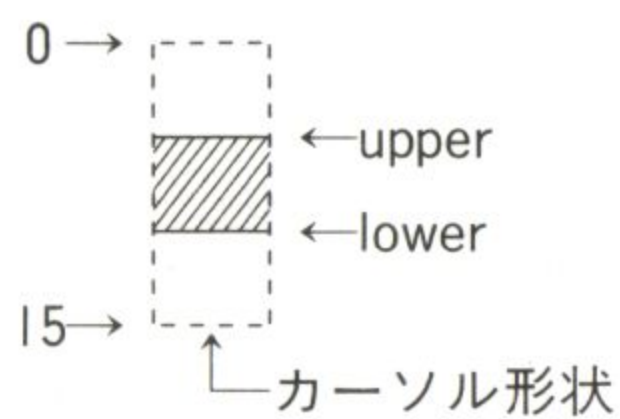
書式

LOCATE [y][,[x][,[flag][,[upper][,lower]]]]

- y … 行位置
- x … 桁位置
- flag … カーソル表示フラグ
- upper … カーソルの上端値
- lower … カーソルの下端値

機能

カーソルを(y, x)で示される位置に移します．y または x を省略すると現在の行または桁位置が使われます．

flag =0とすると，カーソルを表示しません．flag =1とすると，カーソルを表示します．省略すると前の状態を引き継ぎます．

upper と lower はカーソル形状の上端値と下端値です．lower を省略すると，upper と同じ値が採用されます．

例

```
LOCATE ,, 0                 …カーソルを消す.
LOCATE ,,, 8, 15            …カーソル形状を下半分(▄)にする.
LOCATE 2, 10: PRINT  "ABC"
```

LOCK/UNLOCK（他のプロセスからのアクセスの禁止/解除）

書式

```
LOCK [#]n[,[start][TO end]]
      ↑    ↑         ↑
      │    │         └終末レコード
      │    └先頭レコード
      └ファイル番号
UNLOCK [#]n[,[start][TO end]]
```

機能

LOCK はファイルの start～end で示される範囲を他のプロセスに対しアクセス禁止にします．start，end はランダム・ファイルではレコード番号，バイナリ・ファイルでは先頭からのバイト位置です．

UNLOCK は LOCK でロックした範囲を解除します．

start を省略すると 1 とみなされます．TO end を省略すると，start 番号のレコードだけロックされます．

start と end を共に省略すると，ファイル全体をロックします．

シーケンシャル・ファイルでは start と end の指定は意味がなく，いつもファイル全体がロックされます．

LOCK，UNLOCK を機能させるためには，MS-DOS 上で一度 SHARE.EXE を実行しておかなければなりません．

例

```
LOCK     #1, 1 TO 4
LOCK     #1, 5 TO 8
   ⋮
UNLOCK   #1, 1 TO 4
UNLOCK   #1, 5 TO 8
```

……まとめて，UNLOCK #1, 1 TO 8 としてはいけない．LOCK で指定したのと同じレコード番号を使わなければならない．

LOF（ファイル・サイズの取得）　関数

書式

```
LOF(n)
    └ファイル番号
```

機能

ファイル番号 n のファイルのサイズをバイト数で返します．

OPEN COM 文でオープンしたデバイスに LOF を使用すると，出力バッファ内の空きバイト数を返します．

```
TYPE Man
   Namae AS STRING * 16
   Year AS INTEGER
END TYPE
DIM A AS Man
```

と定義されたMan型のレコードをセーブした，ファイル番号1のファイルがあったとき，

MaxRec = LOF(1)/LEN(A)

はファイルのレコード総数を示します．

```
' ---/* ファイル・サイズの取得 */---

TYPE Man
    Namae AS STRING * 16
    Year AS INTEGER
END TYPE
DIM a AS Man

OPEN "b:name.dat" FOR INPUT AS #1

PRINT "ファイルのバイト数  : "; LOF(1)
PRINT "ファイルのレコード数: "; LOF(1) / LEN(a)

CLOSE #1
```

```
ファイルのバイト数   : 72
ファイルのレコード数 : 4
```

LOG （自然対数） 関数

書式 LOG(x)

機能 自然対数 $\log_e x$ を求めます．

LPOS （印刷行バッファ内のプリンタ・ヘッドの現在位置の取得） 関数

書式

```
LPOS(n)
     └─プリンタ番号
```

機能 n番で示されるプリンタ(LPTn:)のヘッドの現在位置を取得します．
LPOS関数はタブ文字を展開しませんので，返す値は必ずしもプリンタヘッドの物理的位置とはなりません．

例

```
' ---/* 印字位置の取得 */---

FOR k = 48 TO 72
    LPRINT CHR$(k)
    IF LPOS(1) > 10 THEN LPRINT ←──印字位置が10桁を越えたら改行
NEXT k
LPRINT
```

```
0123456789
:;<=>?@ABC
DEFGH
```

LPRINT　　　　　　（プリンタへのデータ出力）

書　式

```
LPRINT arg1 {;|, }…
        ↑出力データ
```

機　能

arg1以下のデータをプリンタに出力します．データの区切りに“；”と“，”を用いた場合の違いはPRINTと同じです．

LPRINTはプリンタデバイスLPT1:を使用します．

LPRINT USING（書式制御付きのプリンタへのデータ出力）

書　式

```
LPRINT USING format ; arg1{;|, }…
             ↑         ↑出力データ
             └書式制御文字列
```

機　能

arg1以下のデータをformatで示される書式にしたがってプリンタに出力します．

formatの指定方法はPRINT USINGと同じです．

LSET　　　　　　（ファイル・バッファへの値の設定）

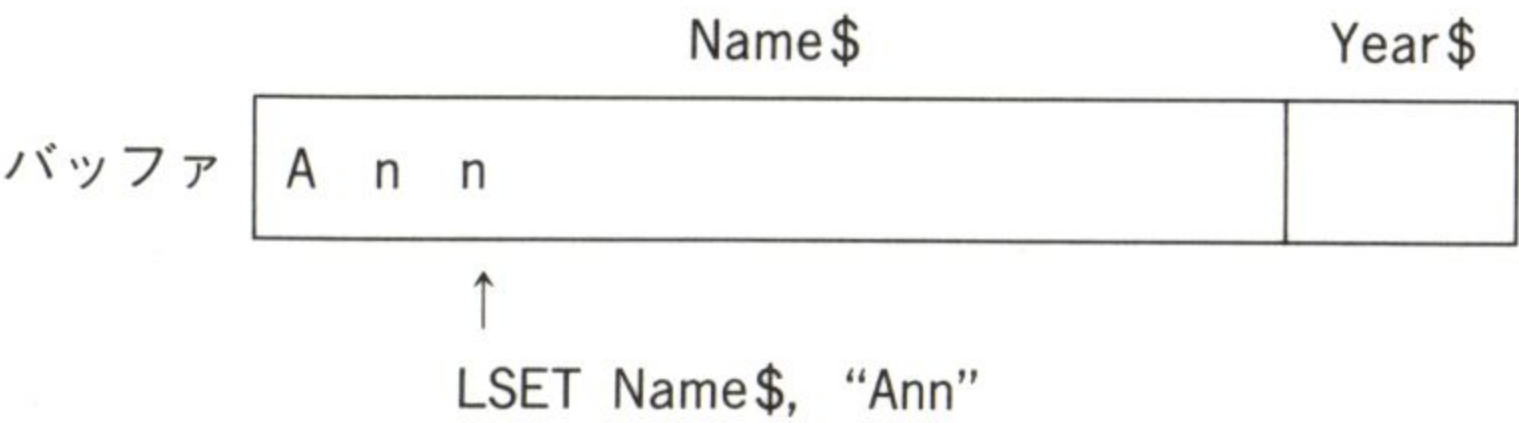
書　式

```
① LSET strvar = strexpression
       ↑          ↑文字列
       └FIELDで設定した文字列変数
② LSET recvar1 = recvar2
       ↑          ↑レコード変数
```

機　能

① FIELD文で設定したリード/ライト・バッファに割り当てられている変数に文字列データを左づめにして格納します．余った部分は空白で埋められます．文字列が長ければ，はみ出した部分は捨てられます．

```
FIELD #1, 16 AS Name$, 2 AS Year$
```

```
                 Name$              Year$
バッファ  | A  n  n               |       |
             ↑
          LSET Name$, "Ann"
```

strvarにはFIELD文で設定した文字列変数ではなく，一般の固定長文字列変数を使用することができます．機能はフィールド・バッファへの設定と同じです．

②異なる型のレコードに対する代入を行います.

```
TYPE S2
  Name$ AS STRING * 2
END TYPE

TYPE S3
  Name$ AS STRING * 3
END TYPE
DIM A AS S2, B AS S3
:
LSET S3 = S2
```

同じ型のレコードに対しては単なる代入文で行えます.

例

MKD$ 参照.

LTRIM$ （左側スペースの削除） 関数

書式

```
LTRIM$(str)
       └─文字列
```

機能

文字列 str の左側にあるスペースを削除します. スペースは, 1バイト文字, 2バイト文字のスペースです.

例

```
A$ ="␣␣␣Hello␣BASIC"
B$ = LTRIM$(A$)          … "Hello␣BASIC"を返す.
```

MID$ （中間文字列の取得） 関数

書式

```
MID$(str, start[, n])
     ↑     ↑      └─取り出す文字数
   文字列   └─取り出す開始位置
```

機能

文字列 str の start 位置から n 個の文字を取り出して返します. n を省略すると文字列の終わりまで取り出します.

日本語を含む文字列には KMID$ を使用してください.

例

```
A$ = "Hello␣BASIC"
B$ = MID$(A$, 7, 3)    … "BAS"が返される.
C$ = MID$(A$, 7)       … "BASIC"が返される.
```

MID$　（文字列の置き換え）

書式

```
MID$(strvar, start[, n])= strexpression
     ↑        ↑      ↑    ↑─置き換える文字列
     │        │      └─置き換える文字数
     │        └─置き換え開始位置
     └─置き換えられる文字列
```

機能

文字列変数strvarのstart位置以後の文字をstrexpressionで置き換えます．nを指定すると，strexpressionの先頭からn文字分を置き換えます．置き換える文字列が長すぎる場合は，はみ出た文字は捨てられます．日本語を含む文字列にはKMID$を使用してください．

例

```
A$ = "This is a pen"
MID$(A$, 11) = "book"
```

…A$は"This is a boo"となる．kは捨てられることに注意．

MKD$/MKI$/MKL$/MKS$　（数値を内部コード形式の文字列に変換）　関数

書式

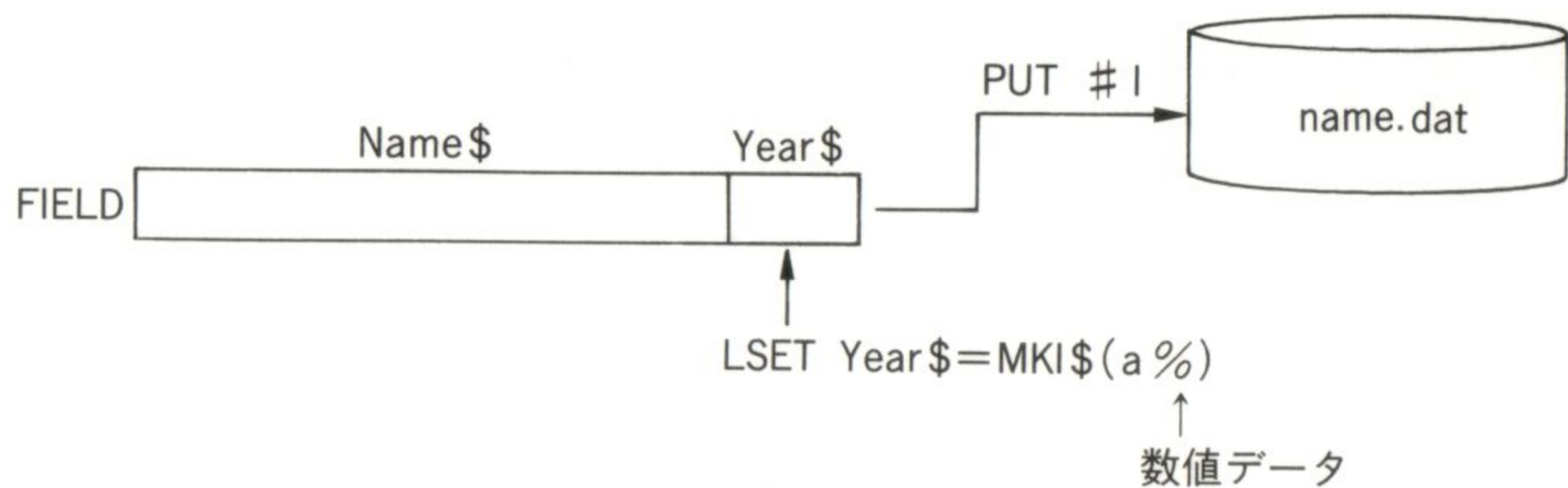
MKD$(double)　←倍精度実数値を8バイト文字列に変換
MKI$(integer)　←整数値を2バイト文字列に変換
MKL$(long)　←倍長整数値を4バイト文字列に変換
MKS$(single)　←単精度実数値を4バイト文字列に変換

機能

FIELDで定義したリード/ライト・バッファは文字列型なので，ここに数値データを格納する場合は，MKD$/MKI$/MKL$/MKS$関数を用いて文字列に変換してから行います．

例

```
' ---/* ランダム・ファイルの書き込み */---

OPEN "b:name.dat" FOR RANDOM AS #1 LEN = 18
FIELD #1, 16 AS Name$, 2 AS Year$

INPUT "Rec No "; Rec
WHILE Rec > 0
    INPUT "name , year "; n$, a%
    LSET Name$ = n$
    LSET Year$ = MKI$(a%)
    PUT #1, Rec
    INPUT "Rec No "; Rec
WEND
CLOSE #1
```

```
Rec No ? 3
name , year ? Ann,19
Rec No ? 1
name , year ? Candy,21
Rec No ? 4
name , year ? Rolla,20
Rec No ? 0
```

MKDIR （サブディレクトリの作成）

書式

MKDIR path
 └サブディレクトリ名

機能

path で示されるサブディレクトリを作成します．

例

MKDIR "B:¥SATO¥WORK"
…ドライブBのサブディレクトリ SATO の下にサブディレクトリ WORK を作る．サブディレクトリ SATO はすでに作成されていなければならない．

MKSMBF$/MKDMBF$ （IEEE 形式の数値を MBF 文字列に変換） 関数

書式

MKSMBF$(single) ←単精度実数値を4バイト文字列に変換
MKDMBF$(double) ←倍精度実数値を8バイト文字列に変換

機能

IEEE 形式の数値を マイクロソフト・バイナリ 形式(MBF)の文字列に変換します．

NAME （ファイル名の変更）

書式

NAME old AS new
 │ └新しいファイル名
 └古いファイル名

機能

old で示されるファイル名を new に変更します．

old と new のパスが異なる場合は，サブディレクトリ間の移動となります．異なるドライブ間の移動はできません．

例

NAME　"abc.bas" AS　"aaa.bas"

… abc.bas を aaa.bas にリネームする．

NAME　"abc.bas" AS　"work¥aaa.bas"

… カレントディレクトリの abc.bas を，サブディレクトリ work に aaa.bas として移動する．カレントディレクトリの abc.bas はなくなる．

OCT$　（数値を8進文字列に変換）　関数

書式

OCT$(n)
　　↑数値

機能

数値 n を 8 進文字列にして返します．

例

OCT$(255)　… 377を返す．

ON ERROR GOTO　（エラー・トラッピングルーチンの設定）

書式

ON ERROR GOTO line
　　　　　　↑トラッピングルーチンの入口

機能

エラー・トラッピングルーチンのエントリポイントを設定します．line はエントリポイントを示すラベルまたは行番号です．

line に 0 を指定すると，エラー・トラッピング処理を停止します．つまりエラーが発生すると，エラーメッセージが表示されプログラムの実行は停止します．

例

```
ON ERROR GOTO ErrorHandler ←──トラッピングルーチンの設定

FOR k = 1 TO 5
    READ x
    PRINT x, LOG(x)
NEXT k
DATA 2,-3,5,-1,6
END
```
（FOR～END：エラーが発生すると Error Handler に移る．）

```
ErrorHandler:
    PRINT x; "Arg is error"
RESUME NEXT ←──エラーのあった文の次の文に移す．
```

```
 2          .6931472
-3          -3 Arg is error
 5          1.609438
-1          -1 Arg is error
 6          1.791759
```

ON event GOSUB（イベント・トラッピングルーチンの設定）

書　式

ON event GOSUB line
　　└イベント　└トラッピングルーチンの入口

機　能

event で示されるイベントが発生したときに分岐するサブルーチンを設定します．line はラベルまたは行番号です．

event として次の 3 つが指定できます．

event	機　　能
COM(n)	n で示される通信ポートに文字を受信したときにイベント・トランピング．n は 1 または 2
KEY(n)	n 番のキーが押されたときにイベント・トラッピング．n は 1 ～25で，対応するキーはファンクション・キー，方向キー，ユーザ定義キー．詳しくは KEY(n)の項を参照
TIMER(n)	n 秒経過したときにイベント・トラッピング n は 1 ～86400秒(24時間)

ON event GOSUB 文は，イベント・トラッピングルーチンを設定するだけです．イベント・トラッピングを許可/禁止/停止するには，次の文で行います．

event ON	イベント・トラッピングを許可します．
event OFF	イベント・トラッピングを行いません．
event STOP	イベント・トラッピングを停止します．停止中に発生したイベントは記憶されています．

ON event 文を含むプログラムをコンパイラ(bc)でコンパイルする際には，/v または/w オプションを指定してください．

例

```
' ---/* 時計の表示 */---

ON TIMER(1) GOSUB Clock
TIMER ON  ←―――イベントトラッピングの許可

CLS
WHILE 1
    PRINT ".";
oWEND
END

Clock:      ' 時間表示ルーチン
    x = POS(0): y = CSRLIN  ' カーソル位置の取得
    LOCATE 1, 70
    PRINT TIME$
    LOCATE y, x             ' カーソル位置の復帰
RETURN
```

（CLS～PRINT "."; ←―――画面に．を表示し続ける）

…画面の右上隅に 1 秒おきに時刻を表示する

ON ～ GOSUB （条件によるサブルーチンコール）

書式

```
ON n GOSUB line1, line2,…
   ↑              ↑―サブルーチンの入口
   └―式
```

機能

式 n の値が 1 なら line1で示されるサブルーチン，2 なら line2で示されるサブルーチン，… へ分岐します．
line は行番号またはラベルです．

例

```
ON n GOSUB    ABC, AAA, XYZ
 ⋮
ABC:
RETURN
AAA:
RETURN
XYZ:
RETURN
```

…n が 1 なら ABC，2 なら AAA，3 なら XYZ のサブルーチンへ分岐する．

ON ～ GOTO （条件による分岐）

書式

```
ON n GOTO line1, line2, …
   ↑            ↑―飛び先
   └―式
```

機能

式 n の値が 1 なら line1，2 なら line2，…へ分岐します．
line は行番号またはラベルです．

例

```
ON n GOTO ABC, AAA, XYZ
```

…n が 1 なら ABC，2 なら AAA，3 なら XYZ へ分岐する．

OPEN　　　　（ファイルのオープン）

書　式

① OPEN filename [FOR mode][ACCESS access][lock] AS [#]n [LEN = reclen]

（filename：ファイル名、mode：ファイルモード、access：アクセスモード、lock：共有モード、n：ファイル番号、reclen：レコード・サイズ）

② OPEN mode,[#]n, filename [, reclen]

（mode：ファイルモード、n：ファイル番号、filename：ファイル名、reclen：レコード・サイズ）

機　能

① filename で示されるファイルをファイル番号 n でオープンします．mode はファイルのオープンモードで以下のものを指定します．

mode	機　　能
OUTPUT	シーケンシャルファイルの出力モード
INPUT	シーケンシャルファイルの入力モード
APPEND	シーケンシャルファイルのアペンドモード
RANDOM	ランダムアクセスモード
BINARY	バイナリモード

access はファイルのリード/ライトアクセスのモードを設定するもので以下のものを指定します．

access	機　　能
READ	読み出し専用モード
WRITE	書き込み専用モード
READ WRITE	読み出し，書き込み両用モード

lockはマルチプロセシング環境で意味を持ち，オープンファイルへの他のプロセスからのアクセスを制限するものです．

lock	機　　　能
指定せず	他のプロセスからアクセスすることはできない
SHARED	他のプロセスから読み出しモードでも書き込みモードでもアクセスできる
LOCK READ	他のプロセスからこのファイルを読み出すことを禁止する
LOCK WRITE	他のプロセスからこのファイルに書き込むことを禁止する
LOCK READ WRITE	他のプロセスからこのファイルに読み出すこと，書き込むことを禁止する

reclenはランダム・ファイルのレコード長(バイト数)を指定します．シーケンシャルファイルでreclenを指定した場合は，リード/ライト用バッファのサイズとなります．デフォルトのバッファサイズは512バイトです．

② filenameで示されるファイルをファイル番号nでオープンします．この書式は他のBASICとの互換性を保つためのものです．
modeはファイルのオープンモードで以下のものを指定します．

mode	機　　　能
O	シーケンシャルファイルの出力モード
I	シーケンシャルファイルの入力モード
A	シーケンシャルファイルのアペンドモード
R	ランダムアクセスモード
B	バイナリモード

reclenの意味は①の書式の場合と同じです．
①，②の書式とも，filenameに次のようなデバイスファイルを指定できます．

KYBD:　**キーボード**
SCRN:　**スクリーン**
COMn:　**通信ポート(RS-232C)**
LPTn:　**プリンタ**
CONS:　**標準出力**

例

```
OPEN "abc.dat" FOR APPEND AS #1
```

…ファイル abc.dat をアペンドモードでオープンする．つまりすでにあるデータの終わりからデータを書き出すモード．


```
TYPE Man
      Namae AS STRING *16
      Year  AS INTEGER
END TYPE
DIM A AS Man
OPEN "abc.rnd" FOR RANDOM AS #1 LEN = LEN(A)
```

…Man 型のデータをレコードとするランダム・ファイルのオープン

```
OPEN "LPT1:BIN" FOR OUTPUT AS #1
```

…プリンタデバイスをバイナリモードの出力モードでオープン

```
OPEN "abc.rnd" FOR RANDOM ACCESS READ LOCK WRITE
                                                    AS #1
```

…abc.rnd をランダム・ファイルの読み出し専用ファイルとしてオープン．このファイルはほかのプロセスからの書き込みを禁止している．

OPEN COM　　（通信ポートのオープン）

書式

```
OPEN "COMn: optlist1, optlist2" [FOR mode] AS[#]n [LEN = reclen]
```


機能

通信ポートのCOM1またはCOM2をオープンします．
optlist1は次の形式です．

[speed][,[parity][,[data][,[stop]]]]

オプション	機　　能
speed	ボーレート．9600, 4800, 2400, 1200, 600, 300, 150, 75
parity	パリティチェック．N…チェックしない，E…偶数パリティ，O…奇数パリティ
data	データビット長．5，6，7，8
stop	ストップビット長．1，2

optlist2に指定できるオプションは以下のとおりで，複数指定するときはカンマ(,) で区切ってください．

オプション	機　　　能
ASC	アスキーモードでオープン ・タブをスペースに展開 ・行末にキャリッジリターンコードを付加 ・[CTRL]+[Z]をファイルの終わりとして処理する ・XON/XOFF 制御を許可する ・通信チャネルをクローズしたときに[CTRL]+[Z]コードを送信
BIN	バイナリモードでオープン ・タブを展開しない ・行末にキャリッジリターンコードを付加しない ・[CTRL]+[Z]をファイルの終わりとして処理しない ・通信チャネルをクローズしたときに[CTRL]+[Z]コードを送信しない
CD[m]	m ミリセカンド過ぎても DCD(Data Carrier Detect)が上位にならないときデバイスのタイムアウトが起こる PC-9801シリーズでは m＝0とする
CS[m]	m ミリセカンド過ぎても CTS (Clear To Send) が上位にならないとき，デバイスのタイムアウトが起こる PC-9801シリーズでは m＝0とする
DS[m]	m ミリセカンド過ぎても DSR(Data Set Ready)が上位にならないときデバイスのタイムアウトが起こる
LF	キャリッジリターン文字(0DH)のあとにラインフィード文字(0AH)を自動的に付け加える
OP[m]	オープンに成功するまでの待ち時間を m ミリセカンドに設定する．m を省略すると待ち時間は無期限となる．OP を指定しないと CD と DS で設定した m の長い方に10倍した時間待つ
RB[n]	受信バッファのサイズを n バイトに設定 デフォルトのサイズは512バイト
RS	RTS(Request To Send)の検知を抑制する

mode は次のものを指定します．

mode	機　　　能
OUTPUT INPUT RANDOM	シーケンシャル出力モード シーケンシャル入力モード ランダムアクセスモード

例

次のプログラムは2台のコンピュータ間でRS-232Cを介してデータの送受信を行うものです.

・送信側プログラム

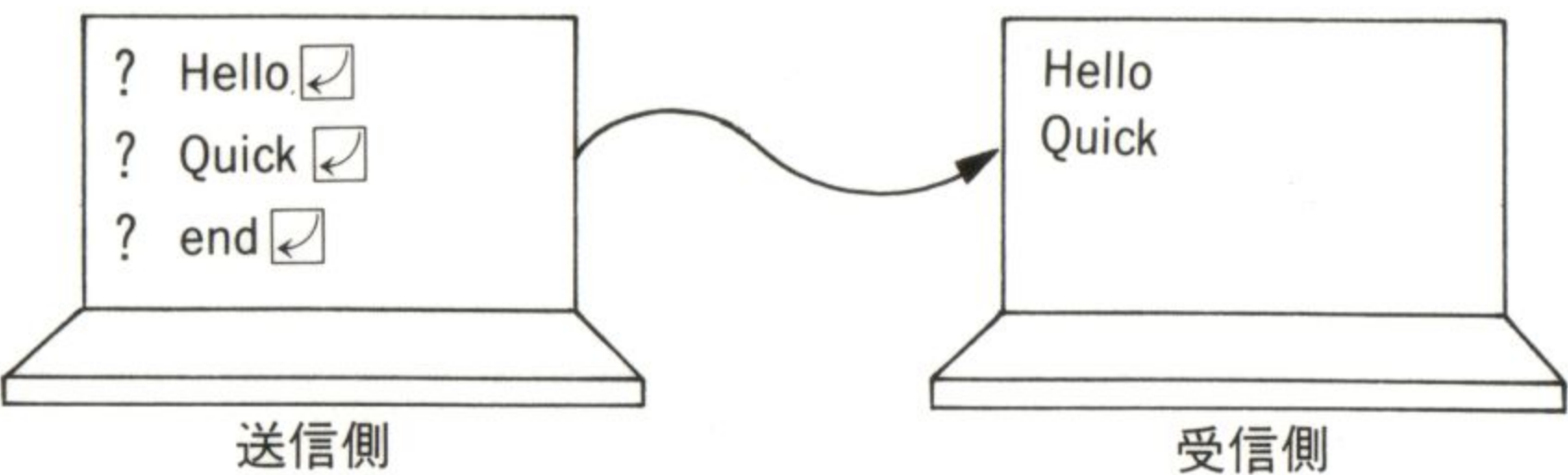
```
OPEN "COM1:9600,N,8,1,ASC" FOR OUTPUT AS #1

DO
    INPUT a$
    PRINT #1, a$
LOOP WHILE a$ <> "end" ←――endという文字列を受信すると終わり
CLOSE #1
```

・受信側プログラム

```
OPEN "COM1:9600,N,8,1,ASC" FOR INPUT AS #1

INPUT #1, a$
DO WHILE a$ <> "end"
    PRINT a$
    INPUT #1, a$
LOOP
CLOSE #1
```

? Hello↵
? Quick↵
? end↵

Hello
Quick

送信側　　受信側

OPTION BASE (配列添字の下限デフォルト値の設定)

書式

OPTION BASE n
　　　　　　　↑添字の下限

機能

配列添字の下限デフォルト値をn(0または1)に指定します. OPTION BASEを行わなければ, 配列添字の下限デフォルト値は0となります.

OPTION BASE文は, モジュール内の配列を宣言する前に一度しか使用できません.

例

```
OPTION BASE 1
DIM A(20)            …A(1)～A(20)の配列が確保される.
```

OUT　（ポートへの1バイト出力）

書式

```
OUT port, data
    ↑        ↑
    │        └出力データ(1バイト)
    └ポート番号(0～65535)
```

機能

port で示されるポートに data で示されるデータを出力します．

例

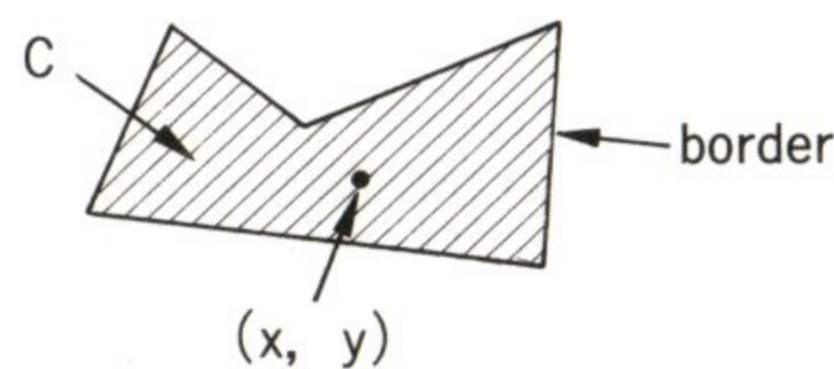
INP 参照．

PAINT　（閉ループ内のぬりつぶし）

書式

```
PAINT [STEP](x, y)[,[c][,[border][, back]]]
               ↑     ↑    ↑          ↑
               │     │    │          └バックグラウンドタイルスライス
               │     │    └図形の境界色
               │     └ぬる色またはタイルパターン
               └ぬりつぶしの開始点
```

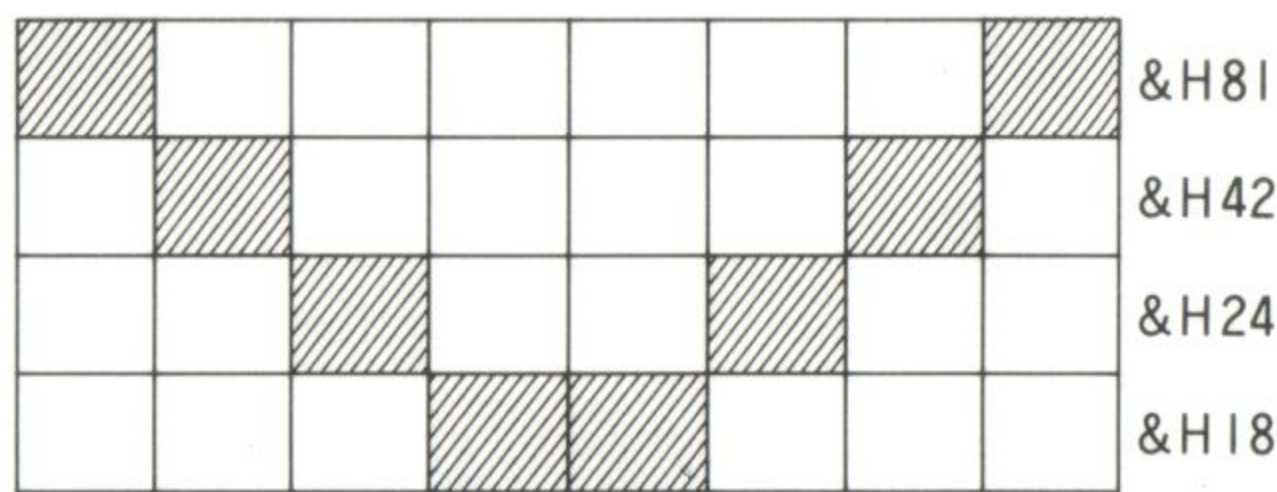
機能

border で示される色(属性番号)の閉ループ内を色 c(属性番号)でぬりつぶします．ぬりつぶしの開始点は(x, y)です．(x, y)の前に STEP を付けると現在位置からの相対距離となります．

c にタイルパターン文字列を指定するとタイリングを行います．タイルパターン文字列は，以下のような8ビット4段のデータが，8ピクセル1行のパターンを示します．

つまり，タイルパターンを構成する文字列データ Tile$は，次のようになります．

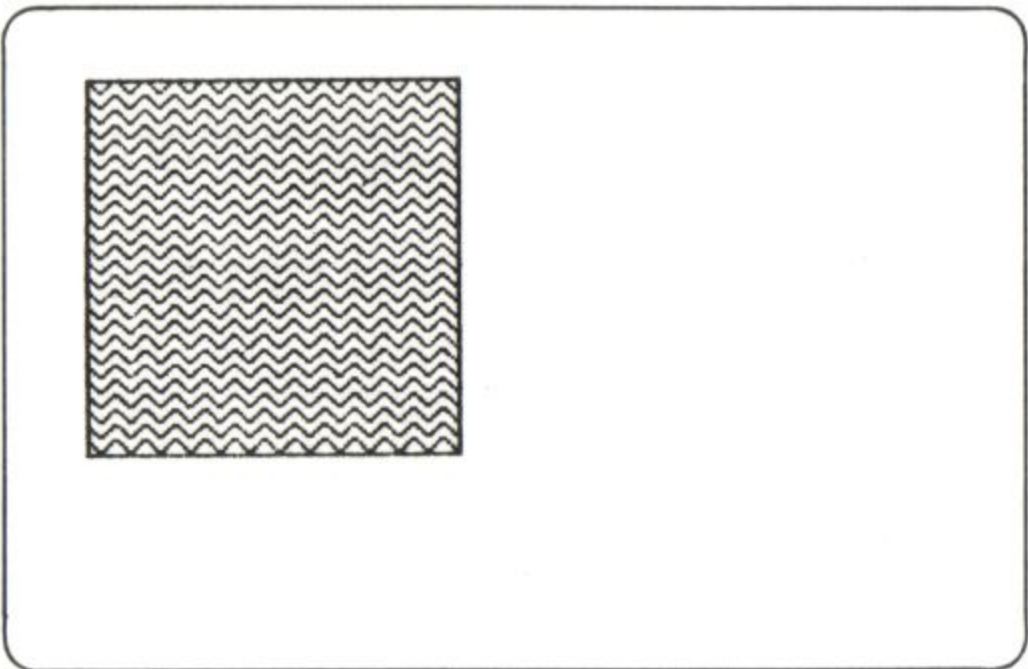
Tile$= CHR$(Blue)+CHR$(Green)+CHR$(Red)+CHR$(intensity)

それでは次のようなパターンを紫色でタイリングするための文字列 Tile$を示します．

&H81
&H42
&H24
&H18

```
Tile$= CHR$(&H81)+CHR$(0)+CHR$(&H81)+CHR$(&HFF)←1段目
      +CHR$(&H42)+CHR$(0)+CHR$(&H42)+CHR$(&HFF)←2段目
      +CHR$(&H24)+CHR$(0)+CHR$(&H24)+CHR$(&HFF)←3段目
      +CHR$(&H18)+CHR$(0)+CHR$(&H18)+CHR$(&HFF)←4段目
           ↑           ↑           ↑           ↑
      Blueのデータ  Greenのデータ  Redのデータ  輝度のデータ
```

例

```
' ---/* タイリング */---

SCREEN 0: CLS

Tile$ = ""
FOR k = 1 TO 16
    READ a$
    a = VAL("&h" + a$)
    Tile$ = Tile$ + CHR$(a)
NEXT k
DATA 81,81,00,ff
DATA 42,42,00,ff
DATA 24,24,00,ff
DATA 18,18,00,ff

LINE (0, 0)-(100, 100), 7, B
PAINT (50, 50), Tile$, 7
```

Redのデータ
輝度のデータ
Greenのデータ
Blueのデータ

PALETTE/PALETTE USING (パレットの設定)

書式

```
① PALETTE   [attribute, color]
              ↑          ↑
              │          └─色番号
              └─属性番号
② PALETTE USING arrayname[(n)]
                    ↑       ↑
                    │       └─配列の要素番号
                    └─配列名
```

機能

① attributeで示される属性番号に，colorで示される色を割り当てます．パラメータを省略すると，attributeとcolorの関係はデフォルト値に設定されます．

②属性番号1～16に対し，配列arrayname(n)，arrayname(n+1) … arrayname(n+15)に格納されている色番号をそれぞれ設定します．添字のnを省略すると，配列の先頭要素から採用されていきます．配列のサイズは，n番目の要素以後に属性番号の数(最大で16)以上なければなりません．

PALETTE文により，属性番号がほかの色番号に変更されると，すでにその属性番号で描画されている図形の色は，新たに設定された色番号の色に一瞬にして変わります．

使用できる色番号はデジタル機とアナログ機，さらにアナグロ機においてはパレット・モードにより異なります．

●デジタル機の場合

色番号，属性番号ともに8種(0～7)です．

0	1	2	3	4	5	6	7
黒	青	緑	水	赤	紫	黄	白

●アナログ機の場合

SCREEN文のパレット・モード1～3に対応して，16色，64色，4096色を使うことができます．属性番号はいずれの場合も16種(0～15)です．

・16色(パレット・モード1)

0	1	2	3	4	5	6	7	8～15
黒	青	緑	水	赤	紫	黄	白	0～7の暗い色

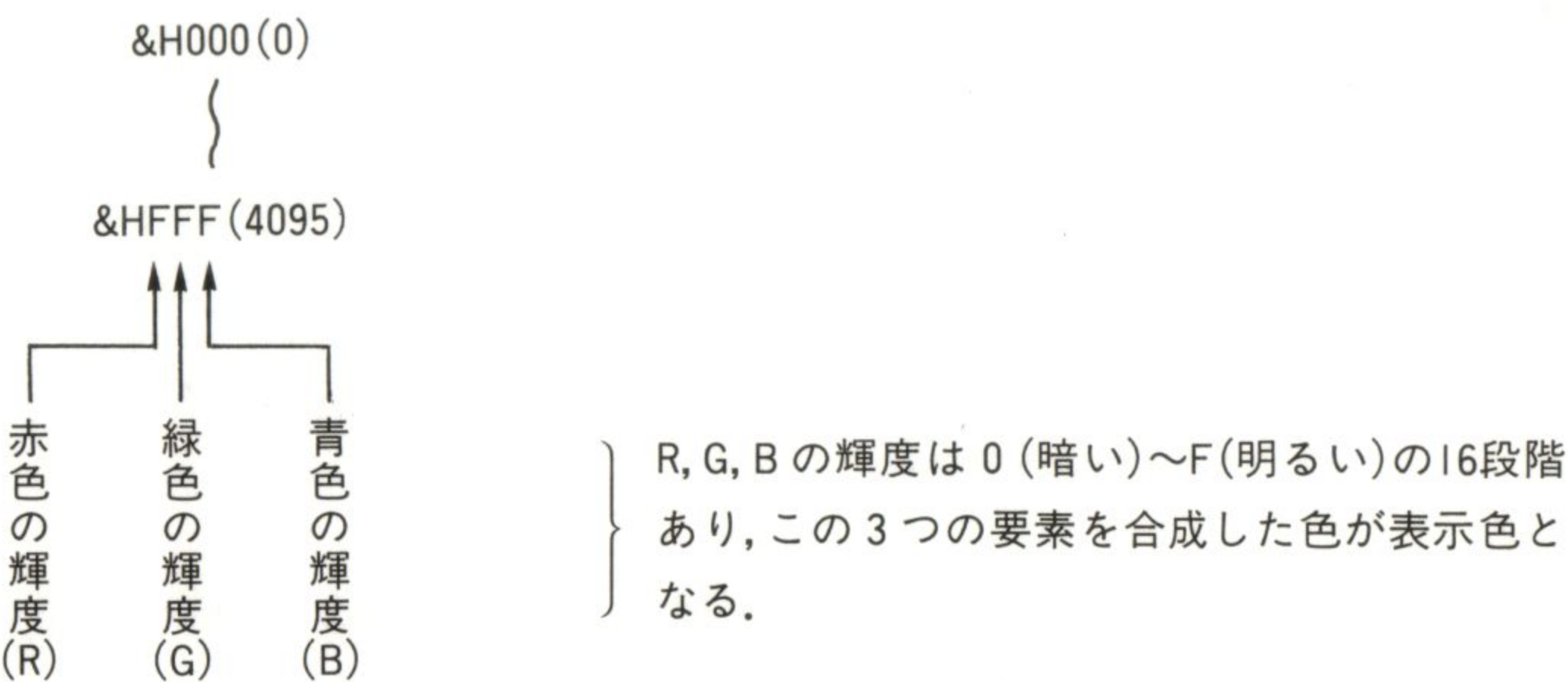
・64色(パレット・モード2)

0〜 7	黒，青，緑，水，赤，紫，黄，白		
8〜15	(0〜7)	＋	薄い青
16〜23	(0〜7)	＋	薄い緑
24〜31	(0〜7)	＋	薄い水色
32〜39	(0〜7)	＋	薄い赤
40〜47	(0〜7)	＋	薄い紫
48〜55	(0〜7)	＋	薄い黄色
56〜63	(0〜7)	＋	薄い白

・4096色(パレット・モード3)

&H000(0)
〜
&HFFF(4095)

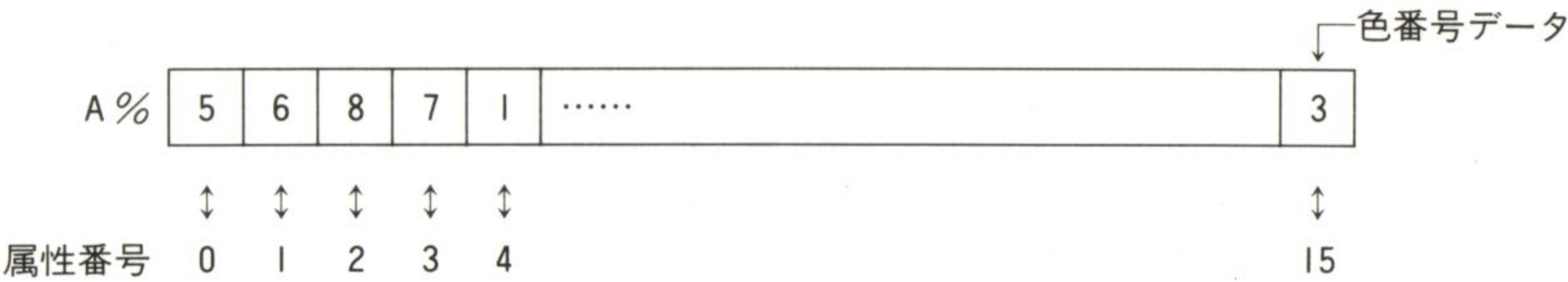
赤色の輝度(R)　緑色の輝度(G)　青色の輝度(B)

R, G, Bの輝度は0(暗い)〜F(明るい)の16段階あり，この3つの要素を合成した色が表示色となる．

例

PALETTE 1, 2　… 属性番号1に色番号2(緑)を割り当てる．すでに属性番号1で描画されている図形は一瞬に緑色に変わる．

PALETTE USING A%

… A%(0)のデータ5が属性番号0に，A%(1)のデータ6が属性番号1に，…　以下同様に割り当てられる．

A%	5	6	8	7	1	……	3 (色番号データ)
	↕	↕	↕	↕	↕		↕
属性番号	0	1	2	3	4		15

```
' ---/* 回るパレット */---

SCREEN 0: CLS

FOR k = 1 TO 7        ' 四角形を描く
    LINE (40 * k, 40)-(40 * k + 40, 80), k, BF
NEXT k

c = 0
DO
    a$ = INKEY$
    IF a$ <> "" THEN        ' キー入力があれば
        FOR i = 1 TO 7
            PALETTE i, ((c + i) MOD 7) + 1 ←――属性番号と色番号の対応を
        NEXT i                                  1つずらす.
        c = c + 1
    END IF
LOOP WHILE a$ <> CHR$(&H1B)    ' ESCキーで終わり

PALETTE ←――――――――――――属性番号と色番号の対応をデフォルト値に設定
```

… このプログラムでは，属性番号に対応する色番号を下図に示すように1つずつシフトさせます．これにより，7つの四角形の色が右から左に移動していくように見えます．

パレット(属性番号)

	1	2	3	4	5	6	7
初期	1(青)	2(　)	3(　)	4(　)	5(　)	6(　)	7(白)
C=0	2	3	4	5	6	7	1
C=1	3	4	5	6	7	1	2

⋮

PCOPY　(スクリーンページを別のページにコピー)

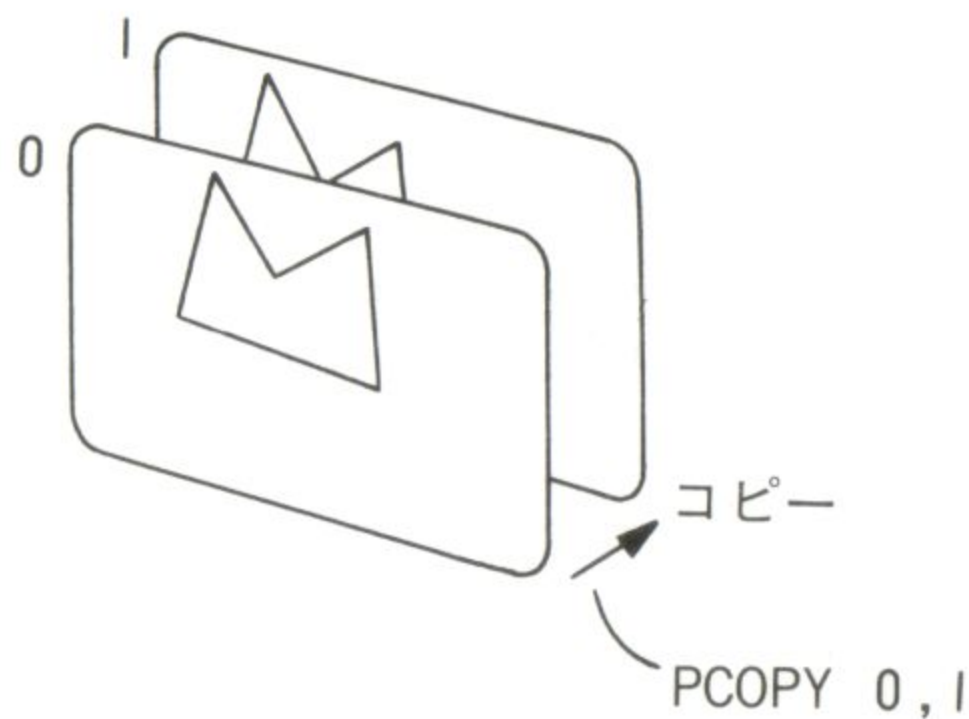
書　式

PCOPY source, dest

source：送り元のページ番号
dest：送り先のページ番号

機　能

source で示される画面ページのデータを，dest で示される画面ページにコピーします．もとの画面ページのデータはそのままです．

0
1
コピー
PCOPY 0,1

PEEK　　（メモリから1バイトをリード）　関数

書式

PEEK(offset)
　　　　└オフセット値

機能

offsetで示されるメモリ・アドレスの1バイトデータをリードします．

offsetはDEF SEG文で設定されている，セグメント・アドレスからのオフセット値です．

例

```
'  テキスト画面の直接アクセス

DEF SEG = &HA000

FOR k = 0 TO 80 * 25 * 2 - 1 STEP 2
    d = PEEK(k)
    IF ASC("A") <= d AND d <= ASC("Z") THEN
        d = d + (ASC("a") - ASC("A"))
        POKE k, d
    END IF
NEXT k
```

… PC-9800シリーズのテキストVRAMは以下のようになっている．

1　2　←桁→　80

行	1	2		80
1	A0000	A0002		A009E
2	A00A0	A00A2		
↕		└アスキー文字の表示は偶数番地だけを使用し，漢字の表示は偶数番地と奇数番地を使用する		
25	A0F80			A0FFE

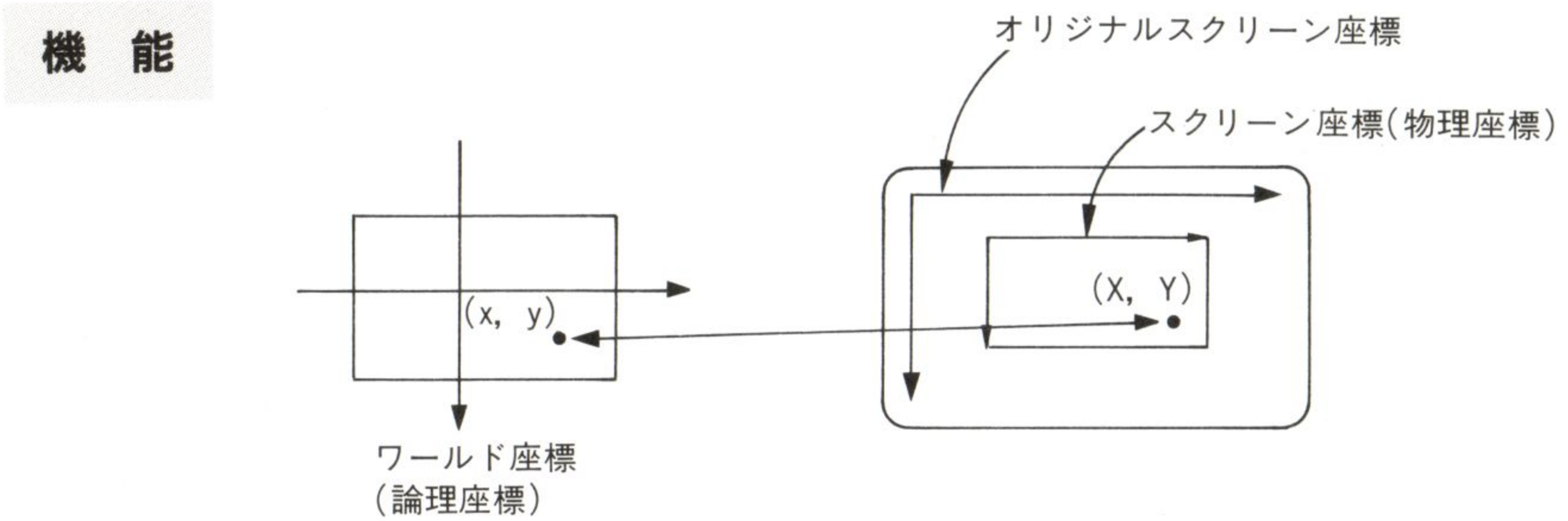
このプログラムは，PEEKでVRAM上のデータをリードし，その文字が大文字なら小文字に変換して，VRAM上に書き直すものです．

PMAP　（論理座標←→物理座標の変換）　関数

書　式

PMAP(position, mode)
　　　　↑座標　　↑変換モード

機　能

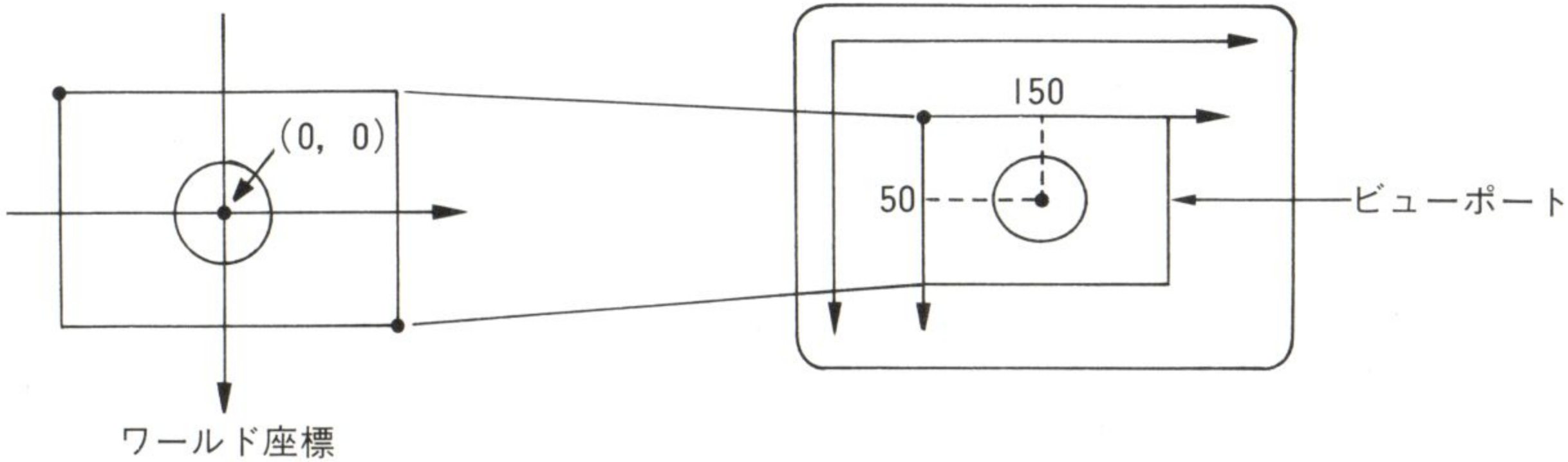
オリジナルスクリーン座標
スクリーン座標(物理座標)
(x, y)
(X, Y)
ワールド座標
(論理座標)

論理座標(x, y)の物理座標上の位置(X, Y)を求めたり，その逆に物理座標(X, Y)の論理座標上の位置(x, y)を求めます．

modeにより次のような変換が行われます．

mode	機　　能
0	論理座標 x →物理座標 X に変換
1	論理座標 y →物理座標 Y に変換
2	物理座標 X →論理座標 x に変換
3	物理座標 Y →論理座標 y に変換

例

```
' ---/* ワールド座標をスクリーン座標に変換 */---

SCREEN 0: CLS
WINDOW (-320, -320)-(320, 320)
VIEW (100, 100)-(400, 300)

LINE (-320, -320)-(320, 320), 1, B
CIRCLE (0, 0), 100, 1

PRINT "スクリーン座標x＝"; PMAP(0, 0)
PRINT "スクリーン座標y＝"; PMAP(0, 1)
```

(0, 0)
150
50
ビューポート
ワールド座標

POINT　（指定位置の色の取得）　関数

書式

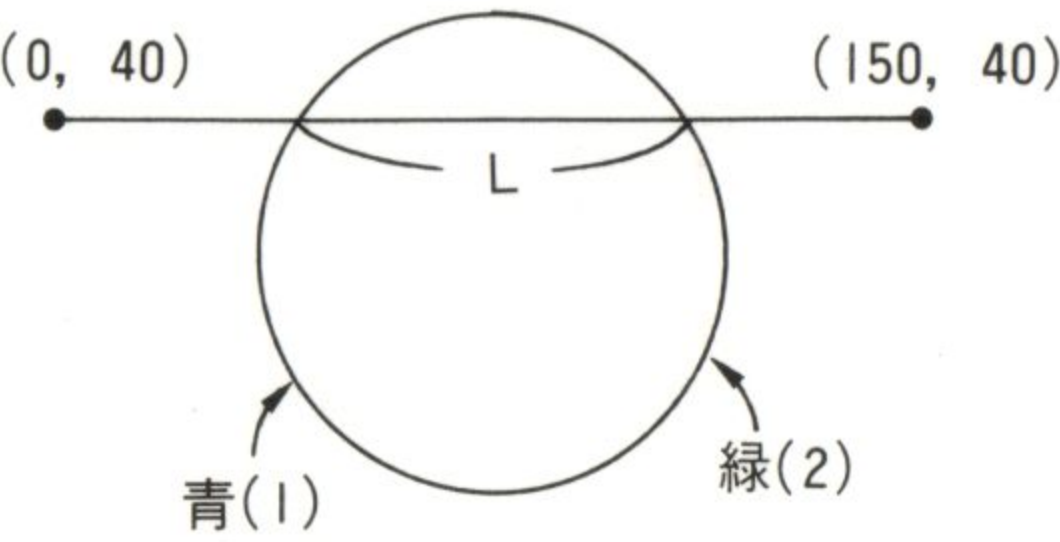
POINT (x, y)
↑—**スクリーン座標**

機能

スクリーン座標(x, y)位置のピクセルの属性番号を返します.

例

```
' ---/* 弦の長さを求める */---

SCREEN 0: CLS
rd = 3.14159 / 180
CIRCLE (100, 50), 40, 1, 90 * rd, 270 * rd ←——青の半円
CIRCLE (100, 50), 40, 2, 270 * rd, 90 * rd ←——緑の半円

FOR x = 0 TO 150
    c = POINT(x, 40)
    IF c = 1 THEN xs = x
    IF c = 2 THEN xe = x
NEXT x
PRINT "Length="; xe - xs ←——————弦の長さ L
```

(0, 40)　(150, 40)
L
青(1)　緑(2)

POINT　（描画現在位置の取得）　関数

書式

POINT (mode)
↑—**取得モード**

機能

グラフィック画面への描画を行ったとき，描き終わった最後の座標は内部的に記憶されています．この座標を描画現在位置またはLP(Last referenced point)位置と呼びます．POINTは，この描画現在位置の座標を取得します．

mode	返される座標
0	物理座標での x
1	物理座標での y
2	論理座標での x
3	論理座標での y

例

```
' ---/* 描画現在位置の取得 */---

SCREEN 0: CLS

PRINT "最初の現在位置  "; POINT(0), POINT(1)

LINE (100, 100)-(200, 200)
PRINT "ｌｉｎｅ後    "; POINT(0), POINT(1)

CIRCLE (150, 150), 100
PRINT "ｃｉｒｃｌｅ後 "; POINT(0), POINT(1)
```

```
最初の現在位置  320        200
ｌｉｎｅ後      200        200
ｃｉｒｃｌｅ後  150        150
```

POKE　（メモリに1バイトをライト）

書式

```
POKE  offset, byte
       ↑        └書き出すデータ
       └オフセット値
```

機能

offsetで示されるメモリ・アドレスに，byteで示される1バイトのデータをライトします．

offsetはDEF SEG文で設定されているセグメント・アドレスからのオフセット値です．

例

PEEK参照．

POS　（現在のカーソル欄の取得）　関数

書式

```
POS(dumy)
    └ダミーの数値引数．通常0を指定
```

機能

現在のカーソル位置の欄座標（x座標）を取得します．

例

CSRLIN参照．

PRESET　（指定点のドットの消去）

書　式

```
PRESET [STEP](x, y)[,c]
              ↑        ↑─表示色
              └─指定点の座標
```

機　能

(x, y)位置に色属性cの色のドットを付けます．cを省略すると，バックグラウンド色が採用されます．したがって，PRESET (x, y)は指定点のドットを消去することになります．

例

PSET参照．

PRINT　（スクリーンへの出力）

書　式

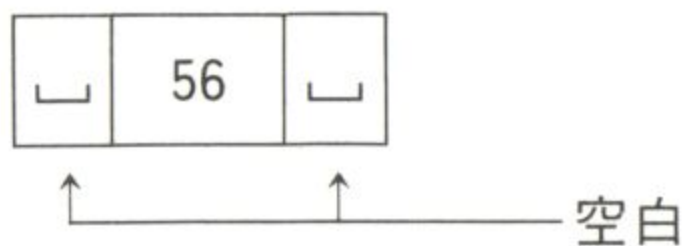
```
PRINT [arg{ , | ; }arg … ]
        ↑─出力データ
```

機　能

argで示されるデータをスクリーンへ出力します．

argが数値データの場合，数字のうしろには必ず空白が付けられ，数字の前には，正の数値の場合は空白，負の数値の場合は負符号(－)が付けられます．

␣	56	␣

↑　↑──空白

出力データが単精度実数および倍精度実数の場合，出力桁数に応じて固定小数点書式と浮動小数点書式の2通りの方法をとります．

●固定小数点書式

7桁以下の単精度実数，16桁以下の倍精度実数は次のような固定小数点書式で表示されます．

・0000011
・0000000000000011

●浮動小数点書式

7桁以上の単精度実数，16桁以上の倍精度実数は次のような浮動小数点書式で表示されます．

1.1E-07
1.1D-16

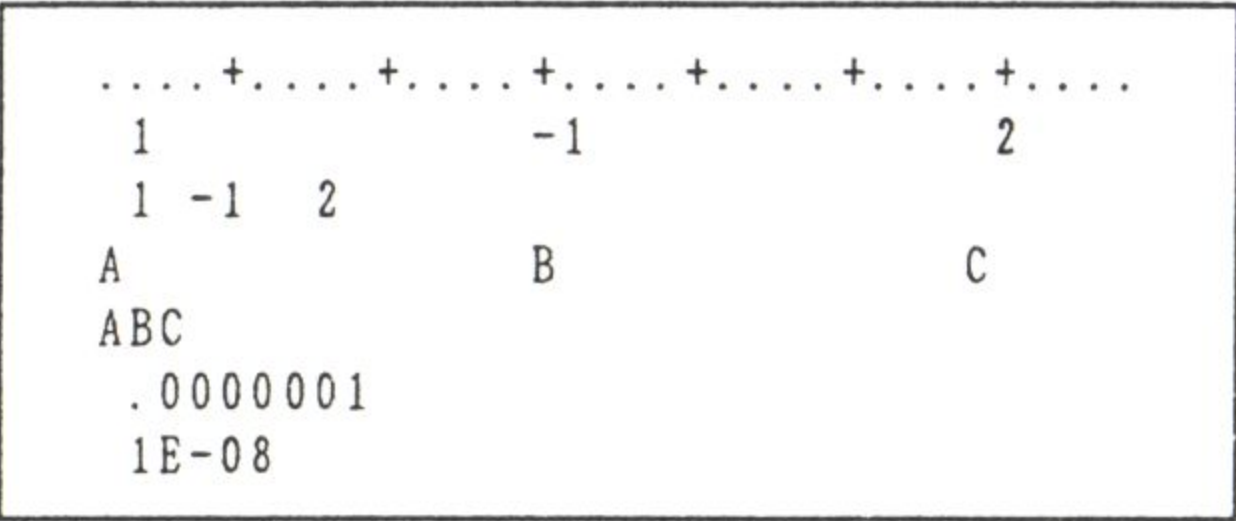
出力データの区切りは“,”と“；”を用い，その意味は以下のとおりです．

　，　…　データの出力幅を14桁単位で区切って出力します．
　；　…　前に出力されたデータの直後に出力します．

PRINT 文の最後に“,”または“；”を置くと，表示位置はその行にとどまります．
引数のない単独の PRINT 文は改行のみを行います．

例

```
PRINT "....+....+....+....+....+....+...."
PRINT 1, -1, 2
PRINT 1; -1; 2
PRINT "A", "B", "C"
PRINT "A"; "B"; "C"
PRINT .0000001
PRINT .0000001 / 10
```

```
....+....+....+....+....+....+....
 1             -1             2
 1 -1  2
A             B             C
ABC
 .0000001
 1E-08
```

PRINT USING　（書式指定付きのスクリーンへの出力）

書　式

PRINT USING　format ; arg {, | ;}…

- format ← 書式制御文字
- arg ← 出力データ

機　能

format で示される書式制御文字にしたがって，arg 以後のデータをスクリーンに出力します．
format 中に指定できる書式制御文字は，以下のとおりです．

	書式制御文字	機　　能
文字列の制御	!	文字列の最初の1文字だけを表示
	& ␣ ␣ … &	&と&の間に置いた空白の数n+2個の文字を表示 文字列がn+2より短いと文字列を左詰めにし，右側は空白となる
	@	可変長の文字列フィールドを意味し，引数の文字列はそのままの幅で表示される
数値の制御	##…. #…	表示桁数の指定．#の個数が整数部，小数部のそれぞれの桁数を示す #で指定した出力桁数より出力データの桁数が少なければ，数値は右詰めにして出力される
	+	数値の前後に，その数値の符号(+または−)を表示させる．+を書式制御文字の先頭に置けば数値の前に +を，書式制御文字列の末尾に置けば数値のあとに符号を表示する
	−	書式制御文字列の末尾に置き，負の数値の前に負符号を表示する
	＊＊	数値フィールドの空白を＊で表示 ＊＊は2桁分の桁数指定として数えられる
	¥¥	数値の直前に円記号を1つ表示．¥¥は2桁分の桁数指定として数えられる
	＊＊¥	数値フィールドの空白を＊で表示し，数値の直前に円記号を表示 ＊＊¥は3桁分の桁数指定として数えられる
	,	整数部データを3桁ごとにカンマ(,)で区切る．カンマは文字列中の小数点の左側に置く ，は1桁分の桁数指定として数えられる
	^^^^ ^^^^^	E+xxのような指数形式を指定する E+xxxのような指数形式を指定する 数値の前後に+または−を指定しないと，小数点の左側の1桁が空白または負符号の表示にあてられる
	_(下線)	下線の右にある文字をそのまま表示する これは書式制御文字を表示するのに使う
	%	表示する数値が，指定した数値フィールドよりも長いと数値の前に%記号を表示

例

```
a = 100.56
b = 1234567
c$ = "Hello BASIC"

PRINT "....+....+....+....+....+...."         ┐
PRINT USING "###.##"; a                      │
PRINT USING "######"; a                      │←①
PRINT USING "####.#"; a                      │
PRINT USING "#.###^^^^"; a                   │
PRINT USING "#.##"; a                        ┘

PRINT
PRINT "....+....+....+....+....+...."         ┐
PRINT USING "+#########"; b                  │
PRINT USING "**#########"; b                 │←②
PRINT USING "¥¥########"; b                  │
PRINT USING "########,"; b                   ┘

PRINT
PRINT "....+....+....+....+....+...."         ┐
PRINT USING "&   &"; c$                      │←③
PRINT USING "!"; c$                          │
PRINT USING "@"; c$                          ┘

PRINT
PRINT "....+....+....+....+....+...."                  ┐
PRINT USING "#####.## #####.# #####."; a; a; a        │
PRINT USING "#####.##"; a; a; a                       │←④
PRINT USING "合計=########"; b                         │
PRINT USING "&   & #######"; c$; b                    ┘
```

```
....+....+....+....+....+....   ┐
100.56                          │
   101                          │ ①
 100.6                          │
0.101E+3                        │
%100.56                         ┘

....+....+....+....+....+....   ┐
  +1234567                      │
***1234567                      │ ②
  ¥1234567                      │
1,234,567                       ┘

....+....+....+....+....+....   ┐
Hello                           │ ③
H                               │
Hello BASIC                     ┘

....+....+....+....+....+....   ┐
  100.56    100.6    101.       │
  100.56   100.56   100.56      │ ④
合計= 1234567                   │
Hello 1234567                   ┘
```

PRINT #/ PRINT USING
(シーケンシャル・ファイルへのライト)

書式

① PRINT #n , arg1 {, | ; }arg2 …

（#n：ファイル番号，arg：出力データ）

② PRINT #n , USING format ; arg1 { , | ; }arg2 …

（#n：ファイル番号，format：書式制御文字，arg：出力データ）

機能

PRINT, PRINT USINGでスクリーンに表示したのと同じイメージでファイル番号nのファイルにライトします.

区切り文字としてカンマ(,)を使用すると, データとデータの間の空白(14桁区切り)もファイルに書き込まれることに注意してください.

シーケンシャル・ファイルとして書き出すには, 区切り記号として“,”も書き出さなければなりません. WRITE # を用いればこのような煩雑さはありません.

例

```
' ---/* シーケンシャル・ファイルの書き込み */---

OPEN "man.dat" FOR OUTPUT AS #1

INPUT "Name, year"; Name$, Year

DO WHILE Year >= 0            ←区切りとして","も書き出す.
    PRINT #1, Name$; ","; Year
    INPUT "Name, Year"; Name$, Year
LOOP
CLOSE #1
```

```
Name, year? Ann,18
Name, Year? Candy,19
Name, Year? Eluza,21
Name, Year? Rolla,20
Name, Year? /,-1

A>dump man.dat

 Dump Version 3.00

00000000  41 6E 6E 2C 20 31 38 20-0D 0A 43 61 6E 64 79 2C  Ann, 18 ..Candy,
00000010  20 31 39 20 0D 0A 45 6C-75 7A 61 2C 20 32 31 20   19 ..Eluza, 21
00000020  0D 0A 52 6F 6C 6C 61 2C-20 32 30 20 0D 0A        ..Rolla, 20 ..
```

PSET （指定点にドットの表示）

書式

PSET [STEP] (x, y)[,c]

- (x, y)：指定点の座標
- c：表示色

機能

(x, y)位置に属性番号 c の色のドットを付けます．c を省略すると，COLOR 文で設定されている表示色が採用されます．

例

```
' ---/* PSET.PRESET */---

SCREEN 0: CLS

FOR x = 0 TO 200
    PSET (x, 50), 2  ←緑色でドットをセットしていく．結果として直線が引ける．
NEXT x

FOR x = 0 TO 200 STEP 5
    PRESET (x, 50)      ←セットされているドットを5つおきにリセットする．
NEXT x                    結果として点線になる．
```

```
-----------------------------------
```

PUT　（保存イメージの描画）

書式

```
PUT [STEP](x, y), var[(n)][, mode]
```

- mode：描画モード
- n：先頭データの要素番号
- var：配列名
- (x, y)：表示域の左上隅の座標

機能

GET で配列 var( ) に取得してあるイメージ・データを，(x, y) を左上隅の座標とする領域に描画します．

n はイメージ・データが格納されている配列要素の先頭番号です．(n) を省略して配列名だけを書くと，配列の先頭要素からのデータが使われます．

mode は描画モードを指定するもので以下の 5 種類があります．mode を省略すると XOR とみなされます．

mode	描画モード
PSET	配列に格納されているイメージ・データをそのまま画面に表示 その領域に前に描かれていたものはなくなる
PRESET	配列に格納されているイメージ・データを反転(たとえば白なら黒)して表示．その領域に前に描かれていたものはなくなる
AND	表示域にすでに描かれているデータとピクセルごとの AND をとって描く
OR	表示域にすでに描かれているデータとピクセルごとの OR をとって描く
XOR	表示域にすでに描かれているデータとピクセルごとの XOR をとって描く 背景色を消さずにイメージ・データを動かす場合に利用する

ピクセルごとの論理演算はプレーンごとに行われます．たとえば，反転(NOT)は次のように行われます．

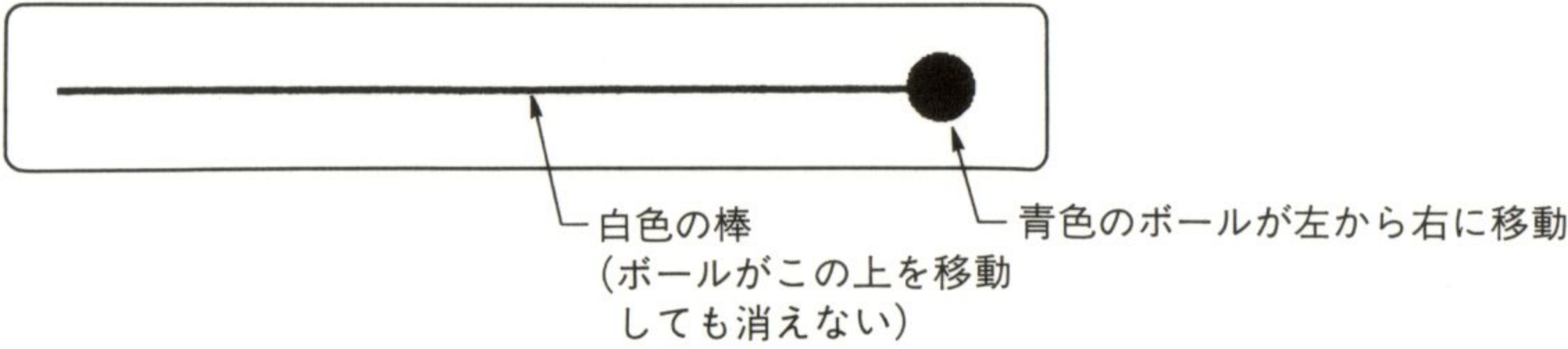
	NOT		
黒 (000)	↔	(111)	白
青 (001)	↔	(110)	黄
緑 (010)	↔	(101)	紫
赤 (100)	↔	(011)	水

↑↑↑
Red | Blue
Green

例

```
' ---/* 背景の中を移動させる */---

SCREEN 0: CLS
x = 45: y = 41
byte = 4 + 4 * ((x + 7) ¥ 8) * y
s = byte ¥ 2 + 1
DIM p%(s)

CIRCLE (20, 20), 15, 1:  ' 円
PAINT (20, 20), 1, 1
GET (0, 0)-(44, 40), p%

CLS
LINE (0, 20)-(400, 22), 7, BF
PUT (0, 0), p%, XOR ←―――――――最初の位置にボールを表示
FOR x = 0 TO 360 STEP 20
    PUT (x, 0), p%, XOR ←――――ボールを消す
    PUT (x + 20, 0), p%, XOR ←―次の位置にボールを表示
    FOR t = 1 TO 500: NEXT t
NEXT x
```

白色の棒
(ボールがこの上を移動
しても消えない)

青色のボールが左から右に移動

PUT　（ランダム・ファイルへのライト）

書式

① PUT [#n][, rec]
　　　↑ファイル番号　↑レコード番号
② PUT [#n],[rec][, var]
　　　　　　　　　　↑ライト・データが格納されている変数

機能

① FIELD 文で設定されているリード/ライト・バッファのデータをファイル番号 n のファイルにライトします．
②変数 var のデータをファイル番号 n のファイルにライトします．var を指定した場合は，FIELD 文でリード/ライト・バッファを設定しないでください．

rec はランダム・ファイルでは読み出すレコード番号，バイナリ・ファイルでは読み出しを始めるバイト位置です．レコード番号，バイト位置の先頭番号は1です．

rec を省略すると，ファイル現在位置(最後に GET/PUT でアクセスしたレコードの次のデータ，または SEEK で指定したデータ位置)のデータをリードします．rec は $1 \sim 2^{31}-1$ (2147483647)の範囲の値を指定します．

例

```
OPEN  "b: name.dat" FOR RANDOM AS #1 LEN=18
FIELD #1, 16 AS Namae$,2 AS Year$
    ⋮
LSET Namae$, "Ann"
LSET Year$, MKI$(19)
PUT #1, Rec                ← FIELD で設定したバッファのデータを
                             レコード番号 Rec としてライト
TYPE Man
   Namae AS STRING * 16
   Year AS INTEGER
END TYPE
DIM A AS Man
OPEN  "b: name.dat" FOR RANDOM AS #1 LEN=LEN(A)
    ⋮
A.Namae="Ann"
A.Year=19
PUT #1, Rec, A             ←レコード型変数 A のデータをレコード番号 Rec
                             としてライト
```

RANDOMIZE（乱数列の初期化）

書式

RANDOMIZE [seed]
　　　　　　↑乱数列の種

機能

seed で指定した値(-32768～32767)を使って，乱数ジェネレータを初期化します．

seed を省略すると，実行時に次のような問い合わせがあります．

新しい乱数列を入力してください(-32768 to 32767)->?

例

```
' ---/* サイコロ */---

RANDOMIZE TIMER ←――――― 乱数列の初期化

FOR k = 1 TO 10
    m = INT(6 * RND + 1) ←――― 1～6の整数乱数を発生
    PRINT m;
NEXT k
PRINT
```

```
 3 3 1 1 6 6 6 1 4 5
```

READ（プログラム中のデータを読む）

書式

READ　var，…
　　　↑データを格納する変数

機能

DATA 文で定義されているプログラム中のデータを変数 var にリードします．

変数とデータの型が一致していなければなりません．また READ 文で指定した変数の数に対しデータが少ない場合は，"Out of DATA"エラーが発生します．

例

```
READ A, B, C$
READ D, E
  :
DATA 5, 10, "Basic", 11, 6
```

…A に 5，B に10，C$に"Basic"，D に11，E に 6 がリードされる．

REDIM　（動的配列の割り当てスペースの変更）

書式

```
REDIM [SHARED] var([lower TO] upper)[AS type], …
```

var：配列名
lower：添字の下限値
upper：添字の上限値
type：配列の型

機能

動的配列のサイズを再度設定し直します．配列の次元を変更することはできません．

例

```
'$DYNAMIC
DIM A(50, 50)
    ⋮
ERASE A            ←Aが割り当てられていた領域を解放し，
REDIM A(100, 100)  ←再び100×100のサイズで割り当て直す．
```

REM　（コメント）

書式

```
① REM comment
② ' comment
```

comment：コメント

機能

コメントを記述します．他のステートメント(文)のうしろにコメントを書く場合，REMを用いると，ステートメントとの間にコロン(:)を必要としますが(')を用いると，コロン(:)は必要ありません．

例

```
Sum = 0:REM 初期化
Sum = 0' 初期化
```

RESET　（すべてのディスク・ファイルのクローズ）

書式

```
RESET
```

機能

すべてのディスク・ファイルをクローズします．そのとき，バッファにあるデータはすべてファイルにライトされます．

RESTORE　　（リードデータ位置の設定）

書　式

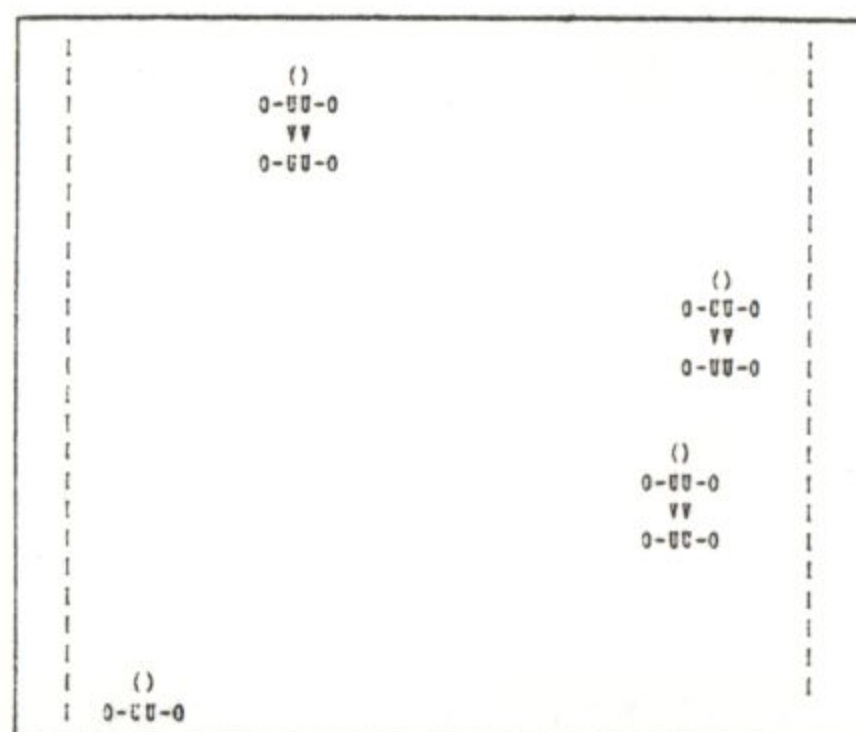
RESTORE [line]
　　　　　└─データのある行

機　能

READ 文が実行されると，DATA 文のデータのリード位置は次々に先に進んでいきます．RESTORE 文により，このリード位置を，指定の行位置のデータに設定することができます．
line は行番号またはラベルです．line を省略すると，プログラム中の最初の DATA 文の先頭データがリード位置となります．

例

```
RESTORE Boy
READ A$, B$          … A$に Jimy，B$に Joe がリードされる．
    ⋮
RESTORE Girl
READ X$, Y$          … X$に Ann，Y$に Candy がリードされる．
    ⋮
Girl:
DATA Ann, Candy, Lisa, Eluza, Nancy
Boy:
DATA Jimy, Joe, Bill, Steave, Tomy
```

```
' ---/* カーレース */---

DATA "  ()  "
DATA "0-UU-0"
DATA "  UU  "
DATA "0-VV-0"
DATA "      "
DATA "      "
DATA "      "

WHILE 1
    RESTORE  ←── READ 文で読むデータをプログラムの最初の DATA 文のデータに設定する．
    TB = INT(30 * RND + 2)  ←─自動車の表示位置を乱数で求める．
    FOR K = 1 TO 7
        READ A$
        PRINT "I"; TAB(TB); A$; TAB(38); "I"
    NEXT K
WEND
END
```

RESUME（エラー・トラッピングルーチンからの復帰）

書　式

① RESUME [0]
② RESUME NEXT
③ RESUME line
　　　　　↑—戻り先の行番号またはラベル

機　能

①エラーを生じた文に戻ります．
②エラーを生じた文の次の文に戻ります．
③ line で指定した行に戻ります．

例

ON ERROR GOTO 参照．

RETURN（サブルーチンからのリターン）

書　式

RETURN [line]
　　　　↑—戻り先の行番号またはラベル

機　能

GOSUB 文または ON event GOSUB 文の次の文に戻ります．line を指定すると，指定された行番号またはラベル位置に戻ります．

RETURN はプロシージャからのリターンには使えません．プロシージャからのリターンには EXIT SUB，EXIT FUNCTION を使います．

例

GOSUB，ON event GOSUB

RIGHT$（右部分文字列の取得）　関数

書　式

RIGHT$(str, n)
　　　↑　　↑—取り出す文字数
　　　└—文字列

機　能

文字列 str の右端から n 文字(バイト)を取り出します．n が文字列の長さより大きければ文字列全体を返し，n が0ならヌル文字列を返します．

例

```
' ---/* 右部分文字列 */---

CLS
p$ = "Basic World !"
FOR k = 1 TO LEN(p$)
    PRINT RIGHT$(p$, k)
NEXT k
```

```
!
 !
d !
ld !
rld !
orld !
World !
 World !
c World !
ic World !
sic World !
asic World !
Basic World !
```

RMDIR　　（サブディレクトリの削除）

書　式

RMDIR path
　　↑—**サブディレクト名**

機　能

path で示されるサブディレクトリを削除します．削除するサブディレクトリ内にはファイルがあってはいけません．

例

KILL　"¥SATO¥WORK ¥＊.＊"　←サブディレクトリ WORK 内のファイルをすべて削除

RMDIR　"¥SATO¥WORK"　←サブディレクトリ WORK の削除

RND　　（乱数の発生）　関数

書　式

RND [(n)]
　　↑—**乱数発生モード**

機　能

0 以上 1 未満の単精度実数型の乱数を 1 つ返します． n の値によって乱数発生モードが異なります．

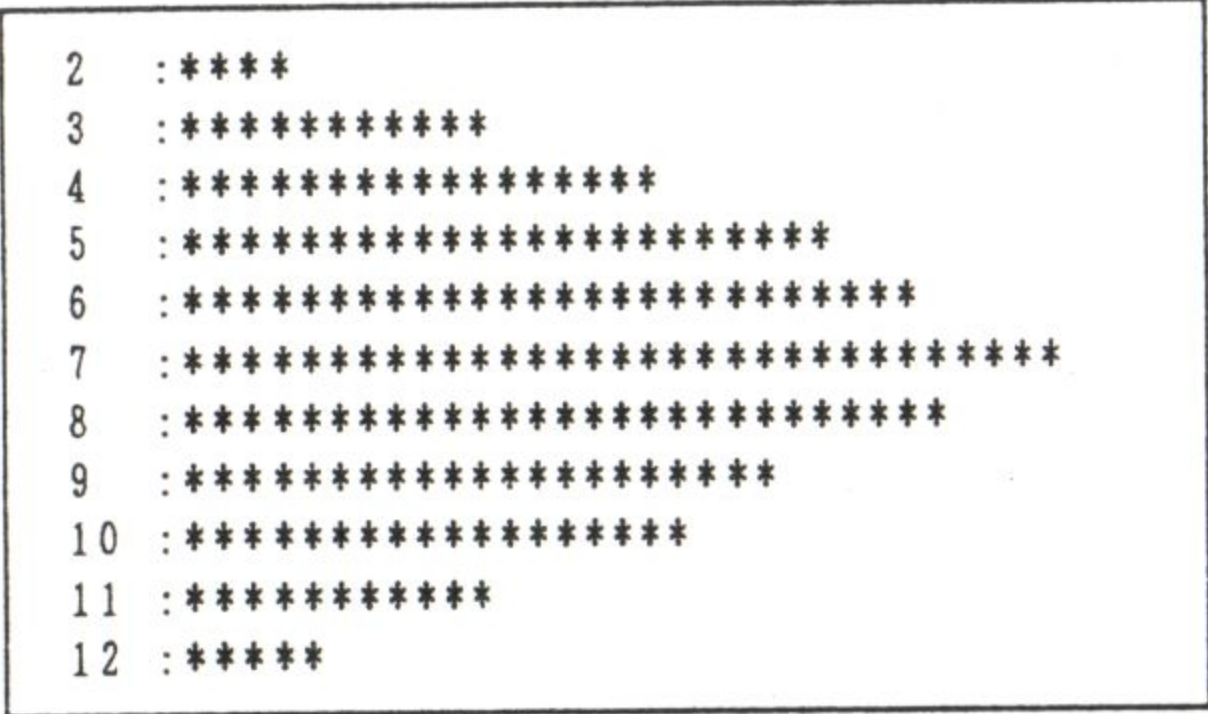
n	機　　能
負	常に同じ値を返す
0	1 つ前の乱数を返す
正または指定せず	次の乱数を返す

n〜m の整数乱数を発生するためには次の式を用います．

INT((m−n＋1) ＊ RND)＋n)

例

```
' ---/* サイコロの目の和の分布 */---

DIM Histo(12)
ERASE Histo

FOR k = 1 TO 4000
    x = INT(6 * RND + 1)      ' 1つ目のサイコロ
    y = INT(6 * RND + 1)      ' 2つ目のサイコロ
    Histo(x + y) = Histo(x + y) + 1
NEXT k

FOR k = 2 TO 12              ' ヒストグラムの表示
    PRINT k; TAB(5); ":";
    FOR j = 1 TO Histo(k) / 20
        PRINT "*";
    NEXT j
    PRINT
NEXT k
```

```
2  :****
3  :***********
4  :*****************
5  :***********************
6  :**************************
7  :******************************
8  :***************************
9  :*********************
10 :******************
11 :***********
12 :*****
```

分布が正規分布になるよう，サイコロをふる回数を4000回としているため，そのままの個数で表示すると画面をはみ出るので20で割っている．

RSET　　（ファイルバッファへの値の設定）

書式

RSET strvar = strexpression

strexpression ── 文字列

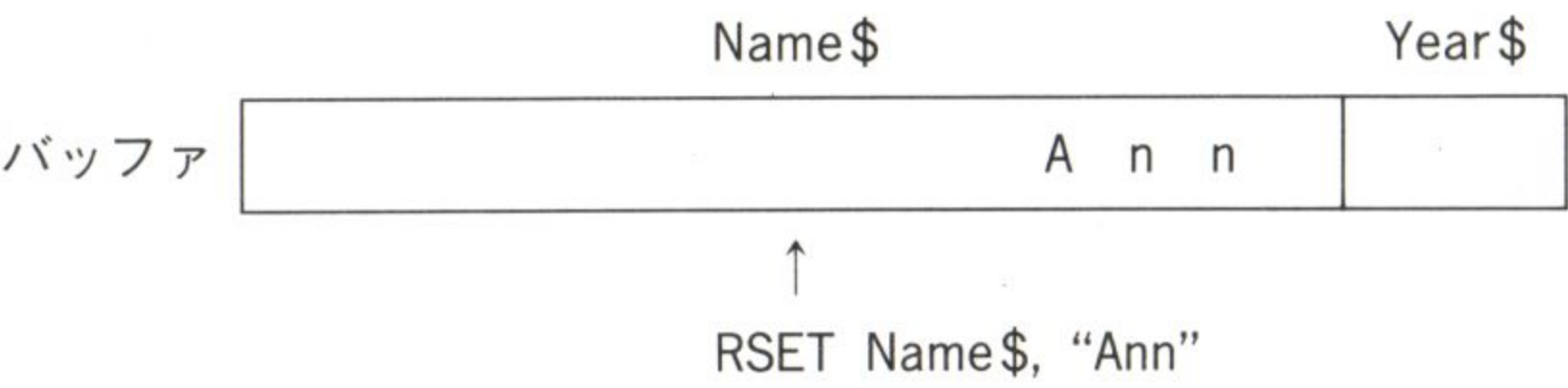
strvar ── FIELDで設定した文字列変数

機能

FIELD文で設定したリード/ライト・バッファに割り当てられている変数に文字列データを右詰めにして格納します．余った部分は空白で埋められます．文字列が長ければ，はみ出した部分は捨てられます．

FIELD #1, 16 AS Name$, 2 AS Year$

	Name$	Year$
バッファ	A n n	

↑ RSET Name$, "Ann"

strvarにはFIELD文で設定した文字列変数ではなく，一般の固定長文字列変数を使用することができます．機能はフィールド・バッファへの設定と同じです．

なお，LSETでは異なるレコード型データ同士の代入が行えましたが，RSETはできません．

例

```
' ---/* 固定長文字列の右詰め */

DIM a AS STRING * 10

a = "Hello" ←―――――― 左詰め
PRINT a

RSET a = "Hello" ←――――― 右詰め
PRINT a
```

```
Hello
     Hello
```

RTRIM$ （右側スペースの削除） 関数

書　式

RTRIM$(str)
　　　　└─ 文字列

機　能

文字列 str の右側にあるスペースを削除します．スペースは1バイト文字，2バイト文字のスペースです．

例

A$ = "Hello␣BASIC␣␣␣␣␣"
B$ = RTRIM$(A$)
　　…"Hello␣BASIC" を返す．

RUN （プログラムの実行）

書　式

① RUN [line]
　　　└─ 実行開始行
② RUN filename
　　　└─ 実行するプログラム名

機　能

①メモリ上のプログラムを line で示される行から再実行します．再実行により変数の内容はクリアされます．line にはラベルを使用できません．line を省略するとプログラムの先頭から再実行します．
② filename で示されるファイルをメモリ上にロードし実行します．QB 環境下で filename のファイルタイプを省略すると，.BAS とみなされ， DOS 環境下で filename のファイルタイプを省略すると，.EXE とみなされます．
　新しいプログラムのロードにより，前のプログラムは消され，オープンファイルはすべてクローズされます．

例

RUN "B:¥WORK¥TEST"
　　…ドライブ B のサブディレクトリ WORK の TEST.BAS が実行される．

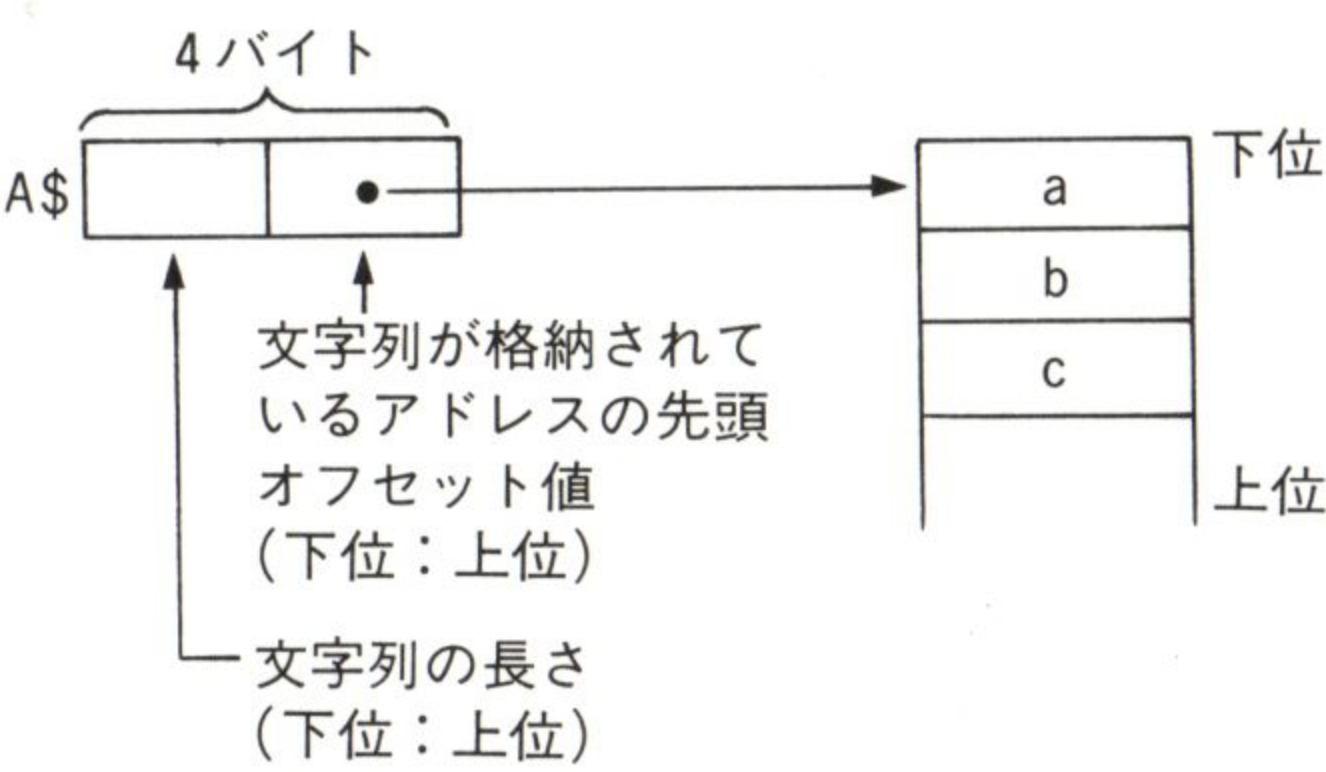
SADD （文字列が格納されているアドレスの取得） 関数

書 式

SADD(strvar)
　　　↑文字列変数

機 能

A$="abc"のような可変長文字列の格納イメージは，次のようになっています．

4バイト
A$
文字列が格納されているアドレスの先頭オフセット値（下位：上位）
文字列の長さ（下位：上位）
a
b
c
下位
上位

つまり，変数A$がある場所と，実際に文字列が格納されている場所が異なるわけです．

SADDは，可変長文字列が実際に格納されているメモリ・アドレスの先頭オフセット値を取得します．SADDは固定長文字列には使用できません．

SADDはBASIC以外の言語で書かれたプロシージャとの間で文字列データを授受する場合によく使います．

例

```
' ---/* 文字列の格納アドレス */---

DECLARE SUB Disp (adr!)

p$ = "Hello!" + CHR$(0)
Disp SADD(p$)

SUB Disp (adr)
    DO
        a = PEEK(adr)
        PRINT CHR$(a)
        adr = adr + 1
    LOOP WHILE a <> 0
END SUB
```

CHR$(0)：文字列の終わりの印
SADD(p$)：先頭アドレス
LOOP WHILE a <> 0：文字列の終わりまでくり返す．

```
H
e
l
l
o
!
```

… 文字列p$の格納アドレスをプロシージャDispに渡し，プロシージャ側では，このアドレスをもとにPEEKで1文字ずつ取り出す．文字列の終わりの印としてヌル文字(CHR$(0))を置くことにする．

SCREEN　（指定位置の文字または色の取得）　関数

書式

SCREEN(y, x[,flag])

- y ─ 行位置
- x ─ 桁位置
- flag ─ 取得モード

機能

flag に真を示す値(0 以外の値)を指定すると，テキスト画面の(x, y)位置のカラー属性(属性番号)を返します．

flag に偽を示す値(0)を指定するか，flag を省略すると，テキスト画面の(x, y)位置の文字コードを返します．

例

```
' ---/* 画面情報の取得 */---

FOR y = 1 TO 25
    FOR x = 1 TO 80
        Col = SCREEN(y, x, 1) ←──── 色の属性番号を取得
        Cha = SCREEN(y, x) ←────── 文字のコードを取得
        IF Col = 7 THEN
            LOCATE y, x       ┐
            COLOR 1           ├←──── 青色の文字に変える
            PRINT CHR$(Cha);  ┘
        END IF
    NEXT x
NEXT y
```

… テキスト画面の(x, y)位置に表示されている文字の ASCII コードと色番号を取得し，もし色が白ならその文字を青色に変える．

SCREEN　（画面モードの設定）

書式

SCREEN [mode][,[pmode][,[apage][, vpage]]]

- mode ─ スクリーンモード
- pmode ─ パレット・モード
- apage ─ アクティブページ
- vpage ─ ディスプレイページ

機能

グラフィック画面のモードを設定します．

mode はスクリーンモードで以下の値を指定します．mode を指定して SCREEN 文を実行すると，テキスト画面，グラフィック画面のすべてが消去されます．

mode	機　　能
81	テキスト
84	640×200　スーパーインポーズ
87	640×400　グラフィック
88, 0	640×400　スーパーインポーズ

pmode はパレット・モードで，使用できる色番号の種類を決めるためのものです．

pmode	機　　能
1	16色
2	64色
3	4096色

apage はアクティブページ(描画データを書き込むページ)，vpage はディスプレイページ(スクリーンに表示されるページ)で，スクリーンモードに応じて次の値を指定します．

mode	ページ番号	
	一般の PC-9801	PC-9801U
81	—	—
84	0～3	1
87	0～1	0
88, 0	0～1	0

例

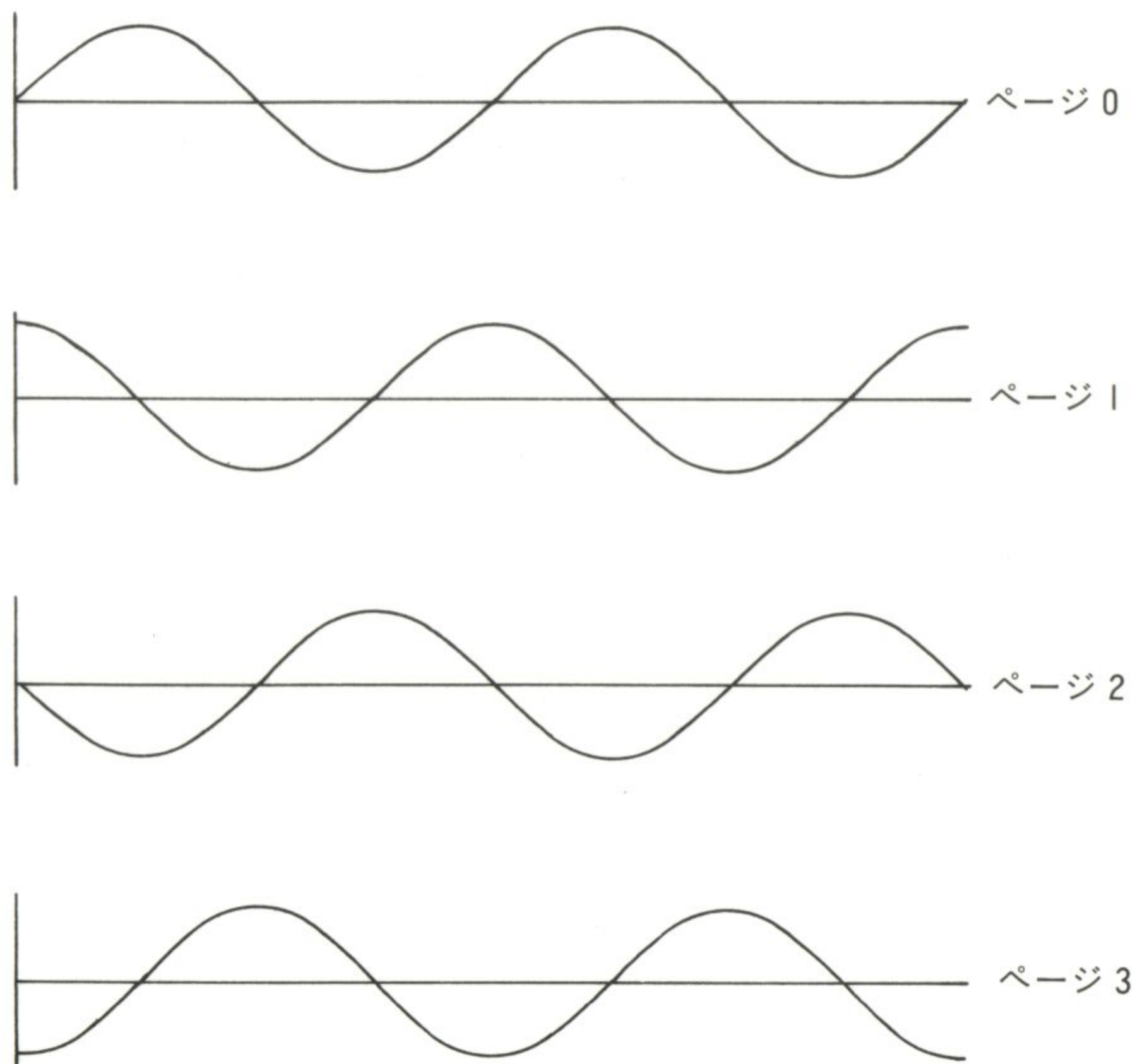
```
' アクティブページとディスプレイページ

FOR k = 0 TO 3
    SCREEN 84, , k, k ←————アクティブページ，ディスプレイページの指定
    CLS
    WINDOW (0, -2)-(720, 2)
    LINE (0, -1.2)-(0, 1.2)      ' 軸線
    LINE (0, 0)-(720, 0)
    FOR x = 0 TO 720 STEP 2                              ┐
        y = SIN((x + 90 * k) * 3.14159 / 180)            │
        IF x = 0 THEN                                    │
            PSET (x, y)          ' 最初の点              │
        ELSE                                             │←——サインカーブを描く
            LINE -(x, y)                                 │
        END IF                                           │
    NEXT x                                               ┘
NEXT k

WHILE 1
    FOR k = 0 TO 3
        SCREEN , , , k ←————ディスプレイページの指定
        FOR t = 1 TO 1000: NEXT t
    NEXT k
WEND
```

… サインカーブの位相を90°ずつずらして4画面に描き，ディスプレイページを逐次切り換えることにより，サインカーブが動いているように見える．

ページ 0

ページ 1

ページ 2

ページ 3

SEEK　　（ファイル現在位置の取得）　関数

書　式

SEEK(n)
　　└ファイル番号

機　能

ファイル番号 n のファイルのファイル現在位置を取得します．取得される値はランダム・ファイルにおいてはレコード番号，シーケンシャルファイルにおいては先頭からのバイト数となります．

例

```
' ---/* ファイル現在位置 */---

OPEN "abc.dat" FOR BINARY AS #1

DO WHILE NOT EOF(1)
    p = SEEK(1) ←―――――ファイル現在位置の取得
    a$ = INPUT$(1, #1) ←――――― 1 文字入力
    IF "a" <= a$ AND a$ <= "z" THEN
        PRINT p; ":"; a$
    END IF
LOOP
CLOSE #1
```

```
1  :a
5  :a
17 :t
31 :u
48 :a
49 :b
50 :c
52 :d
53 :a
54 :t
```

… ファイルから 1 文字単位でリードし，その文字が小文字なら，ファイル先頭からの位置を表示する．

SEEK #　（ファイル現在位置の取得）　関数

書　式

```
SEEK [#]n, pos
       ↑    ↑─移動位置
       └─ファイル番号
```

機　能

ファイル番号nのファイルの，ファイル現在位置をpos位置に移動します．posの値は，ランダム・ファイルにおいてはレコード番号，シーケンシャル・ファイルにおいては先頭からのバイト数となります．レコード番号，バイト数の先頭値はともに1です．

例

```
' ---/* ファイル現在位置の移動 */---

OPEN "abc.dat" FOR INPUT AS #1

SEEK #1, 50
DO WHILE NOT EOF(1)
    a$ = INPUT$(1, #1)
    IF a$ <> CHR$(&HD) THEN PRINT a$;
LOOP
CLOSE #1
```

（DO WHILE～IF の行について）1文字単位でファイルからリード．行末には0DH，0AHが置かれているので2つを出力すると改行が2度行われるため，0DHは出力しない．

… ファイル先頭から50バイトを読み飛ばし，それ以後を1文字ずつ読んでいく．

SELECT CASE　（複数条件判断）

書　式

```
SELECT CASE test
            ↑─被比較式
   CASE p-1 ←─比較式
     block-1    ←実行ブロック
   [CASE p-2
     block-2]
       ⋮
   [CASE ELSE
     block-n]
END SELECT
```

機　能

testとp-1,p-2 …とを比較し，条件を満たしたところのblockを実行し，SELECT CASE文から抜けます．もし条件を満たさなければ，CASE ELSE節に置かれているblock-nを実行します．

blockには複数の文を置くことができます．

testには変数を含む数式または文字列式を書きます．p-1,p-2，…には次の書式のいずれかを書きます．1つのCASE節にこれらの比較式をカンマ(,)で区切って複数指定することもできます．

式1，式2，…

式1 TO 式2

$$\text{IS}\left\{\begin{array}{l} < \\ <= \\ > \\ >= \\ <> \\ = \end{array}\right\}\text{式}$$

例

```
INPUT x
SELECT CASE x
        CASE 1
            [xが1のときの処理]
        CASE 2 TO 5
            [xが2～5のときの処理]
        CASE 6, 8, 9
            [xが6か8か9のときの処理]
        CASE IS>=10
            [xが10以上のときの処理]
END SELECT
```

CASE 2 TO 5 ← 左の値は右の値より小さくなければいけない．

CASE 6, 8, 9 ← カンマ(,)で複数の式を並べて書ける．この場合はOR条件と考えてよい．

CASE IS>=10 ← x IS＞＝10という意味に考えればよい．

```
INPUT Test$
SELECT CASE Test$
        CASE "A" TO "AZZZZZZZZ"
            [Test$がAで始まる文字のときの処理]
        CASE IS < "A"
            [Test$がアルファベット以外の文字のときの処理]
        CASE "Quick"
            [Test$が"Quick"のときの処理]
END SELECT
```

SETMEM　（far ヒープで使用するメモリ量の設定）　関数

書式

SETMEM(n)
　　　　↑増減するバイト数

機能

far ヒープメモリのサイズを n バイト増減します．n に負の値を用いると far ヒープのメモリサイズを減少させます．

BASIC では，プログラム起動時に far ヒープは最大にとられているので，最初から SETMEM で far ヒープを拡大することはできません．

SETMEM は増減後の far ヒープのバイト数を返します．したがって n に0を指定すると，現在の far ヒープのバイト数がわかります．

例

```
PRINT "現在のヒープ・サイズ="; SETMEM(0)
a = SETMEM(-1024)
PRINT "変更後のヒープ・サイズ="; a
```

```
現在のヒープ・サイズ= 178720
変更後のヒープ・サイズ= 177696
```

SGN　（正・負符号の取得）　関数

書式

SGN(x)

機能

x ＞ 0 のとき1，x ＝ 0 のとき0，x ＜ 0 のとき−1をそれぞれ返します．

例

```
D=B＊B−4＊A＊C
ON SGN(D)＋2 GOSUB KYOKON, JYUKON, JITSUKON
```

… 判別式 D の符号に応じて KYOKON（虚根），JYUKON（重根），JITSUKON（実根）へ分岐する．

SHARED　（プロシージャ間の変数の共有）

書式

```
SHARED var [AS type], …
       ↑         ↑—変数の型
       └—変数
```

機能

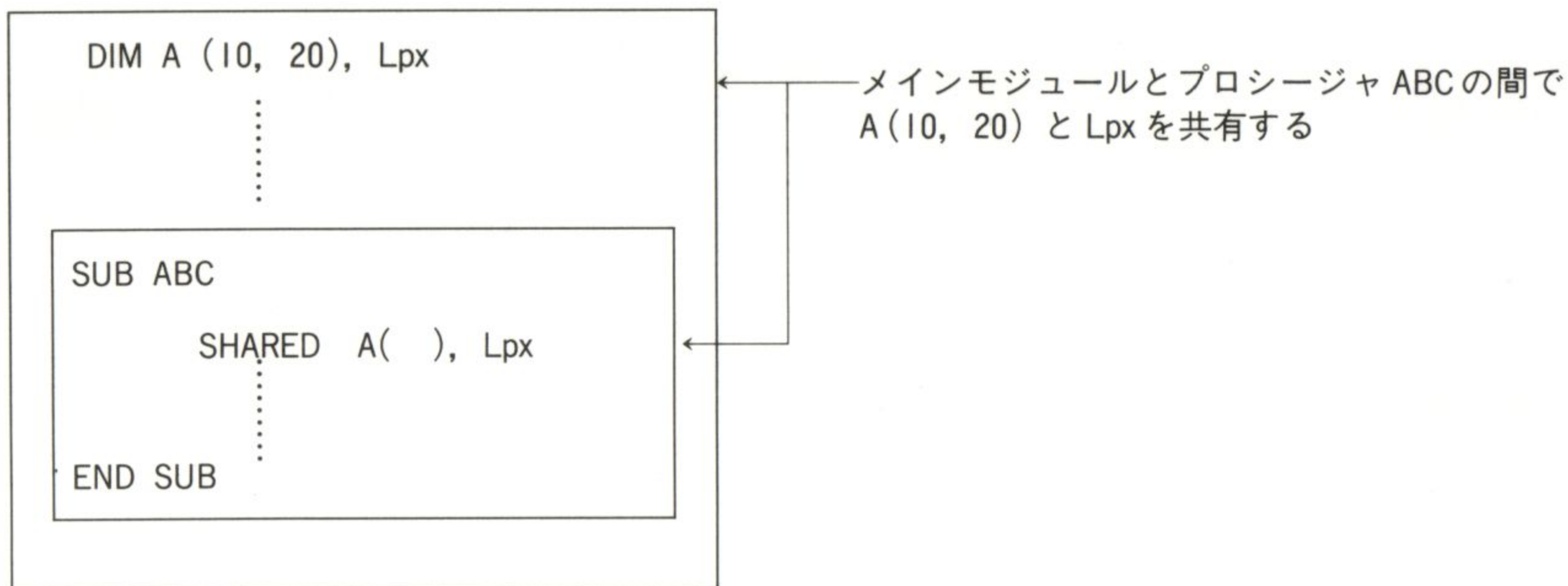
メインモジュールで宣言または使用されている変数を，SHARED 文を行ったプロシージャで共有します．配列の場合は，( ) を付けます．

SHARED 文を置くことができるのはプロシージャの中だけです．

COMMON 文または DIM 文に SHARED 属性を指定すれば，その変数は全プロシージャで共有されますが，SHARED 文で指定した変数は，その SHARED 文が置かれているプロシージャだけで共有されます．

別個にコンパイルしてリンクしたプロシージャとの間では SHARED 文は効果がありません．

例

```
DIM A (10, 20), Lpx          ←— メインモジュールとプロシージャ ABC の間で
   :                             A(10, 20) と Lpx を共有する
SUB ABC
      SHARED  A(  ), Lpx     ←—
   :
END SUB
```

SHELL（実行中のプログラムを一時中断して他のプログラムを実行）

書式

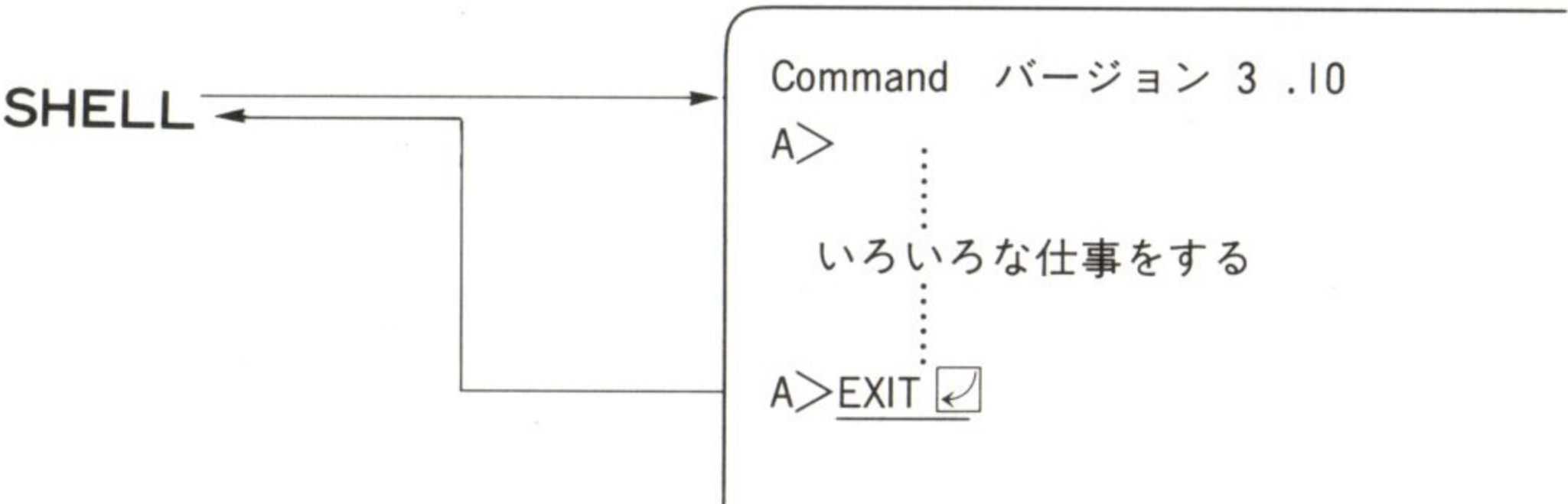
```
SHELL [commandstring]
        ↑—実行するプログラムのコマンドライン文字列
```

機能

BASIC プログラム内からほかのプログラム(これを子プロセスと呼ぶ)を実行し，実行終了後，SHELL 文の次の文に戻ります．

commandstring は実行プログラム名と，それに渡されるパラメータからなる文字列です．実行プログラムのファイルタイプを省略すると，.COM, .EXE, .BAT の順で探します．

commandstring を省略すると，COMMAND.COM が起動されます．この場合，プログラムに戻るには A>EXIT ↵ とします．

```
SHELL ──→  Command  バージョン 3 .10
  ↑        A>
  │          ：
  │        いろいろな仕事をする
  │          ：
  └──────  A>EXIT ↵
```

例

SHELL　"TYPE B: ABC.BAS"

…TYPE コマンドを実行する.

SHELL　"DIR B:*.BAS > DIRFILE"

…DIR をとった結果をI/O リダイレクトして，ファイル DIRFILE に書き込む.

SIN　(サイン)　関数

書式

SIN(x)

機能

xのサインの値を求めます．xの単位はラジアンです．π(3.14159)ラジアンが180°に相当しますから，A°はA * 3.14159/180［ラジアン］となります．

例

```
' ---/* 三角関数 */---

rd = 3.14159265# / 180
PRINT "      x"; TAB(23); "sin(x)"; TAB(43); "cos(x)"; TAB(63); "tan(x)"
FOR x = 0 TO 90 STEP 10
    PRINT USING "####.#  "; x;
    PRINT USING "#############.######"; SIN(x * rd); COS(x * rd); TAN(x * rd)
NEXT x
```

```
   x            sin(x)        cos(x)             tan(x)
  0.0         0.000000      1.000000           0.000000
 10.0         0.173648      0.984808           0.176327
 20.0         0.342020      0.939693           0.363970
 30.0         0.500000      0.866025           0.577350
 40.0         0.642788      0.766044           0.839100
 50.0         0.766044      0.642788           1.191754
 60.0         0.866025      0.500000           1.732051
 70.0         0.939693      0.342020           2.747477
 80.0         0.984808      0.173648           5.671280
 90.0         1.000000     -0.000000   -22877334.000000
```

SPACE$ （スペースの生成） 関数

書式　SPACE$(n)
　　　　　　　　↑— スペースの数

機能　n個のスペースからなる文字列を生成します．

例

```
' ---/* 任意の大きさの四角を描く */---

INPUT "縦，横 "; m, n

PRINT STRING$(n, "=")  ←——=をn個作る

FOR k = 1 TO m
    PRINT "I" + SPACE$(n - 2) + "I"
NEXT k

PRINT STRING$(n, "=")
```

```
縦，横 ? 5，8
========
I      I
I      I
I      I
I      I
I      I
========
```

SPC （スペースの出力） 関数

書式　SPC(n)
　　　　　↑— スペースの数

機能　PRINTまたはLPRINT文と共に使用し，スクリーンまたはプリンタにn個のスペースを出力します．

例

```
PRINT "....+....+....+....+...."
PRINT "Hello"; SPC(5); "Basic"
```

```
....+....+....+....+....
Hello     Basic
```

SQR （ルート：平方根） 関数

書式　SQR(x)

機能　$\sqrt{x}$を求めます．xは正の値を指定してください．

STATIC　（静的変数の宣言）

書　式

```
STATIC var [AS type], …
       └─変数   └─変数の型
```

機　能

変数を静的変数として宣言します．静的変数は，それが宣言されているプロシージャまたはDEF FN関数でローカルです．DEF FN関数内の変数はデフォルトでグローバルです．これをローカルにしたいときにSTATIC宣言を行います．また，自動変数のようにプロシージャのコール/リターンのたびにスタック上に生成／消滅をくり返さず，メモリ上の同じ領域にプログラムの開始から終了まで割り当てられていますから，変数の値は消滅せず保存されています．

STATIC文を置くことができるのは，プロシージャとDEF FN関数の中だけです．

SUB文，FUNCTION文にSTATIC属性を指定すると，プロシージャ内のすべての変数は静的変数となりますが，SHARED宣言されている変数はこの限りではありません．これに対し，STATIC文は指定した変数を静的変数にするもので，SHARED宣言されている共有変数を無効にすることもできます．

STOP　（実行の停止）

書　式

```
STOP
```

機　能

プログラムの実行を停止します．

QB環境下でSTOP文が実行されると，QB画面に戻ります．.EXEファイル中のSTOP文が実行されると，すべてのファイルをクローズしてOSに戻ります．

コンパイラBCで/d,/e,/xオプションを指定していると，STOP文が実行された行番号が表示されます．STOP文に行番号が付いていなければ最も近い位置の行番号が表示され，行番号がどこにもなければ0が表示されます．

STR$ （数値を10進文字列に変換） 関数

書式

```
STR$(n)
    ↑数値
```

機能

数値 n を10進文字列に変換します．正の値なら先頭に空白が 1 つ付けられます．

例

```
PRINT STR$(123)
PRINT STR$(-123)
PRINT STR$(100.05)
PRINT STR$(&HFF)
```

```
 123
-123
 100.05
 255
```

STRING$ （文字列の生成） 関数

書式

```
① STRING$(n, str)
          │   ↑文字列
          └くり返し個数
② STRING$(n, numeric)
          │      ↑ASCII コードまたはシフト JIS コード
          └くり返し個数
```

機能

①文字列 str の先頭文字(1 バイト文字または 2 バイト文字)を n 個くり返した文字列を返します．
② numeric で示される ASCII コード，またはシフト JIS コードに対応した文字を n 個くり返した文字列を返します．

例

SPACE$参照．

SUB （サブルーチン・プロシージャの定義）

書式

```
SUB name[(var [AS type],…)][STATIC]
         │  │      ↑仮引数の型
  ⋮      │  └仮引数
         └プロシージャ名
END SUB
```

機能

nameで示されるサブルーチン・プロシージャを定義します．( ) の中に仮引数を書きます．仮引数がなければ( ) を省略できます．

STATICを指定すると，関数プロシージャ内のローカル変数はすべて(SHARED宣言されている変数は除く)静的変数となります．静的変数については第4章4-2の8を参照．

例

```
' ---/* サブルーチン・プロシージャ */---

DECLARE SUB Disp (Moji$, Num!, Col!) ←── サブルーチン・プロシージャの宣言

CLS                                 ──── くり返し回数
Disp "Hello Quick Basic", 2, 3
Disp "Hello Quick C", 3, 5          ──── 表示色

END

SUB Disp (Moji$, Num, Col)
    COLOR Col
    FOR k = 1 TO Num                ←── サブルーチン・プロシージャの定義．
        PRINT Moji$                     Moji$の文字列をNum回，色Colで
    NEXT k                              表示．
END SUB
```

SWAP　(変数の内容を交換)

書式

SWAP var1, var2
　↑──↑交換する変数

機能

2つの変数var1とvar2のデータを交換します．2つの変数の型は一致していなければなりません．

例

```
A = 10 : B = 20
SWAP A,B              …Aが20，Bが10となる．
```

SYSTEM　(プログラムを終了しOSに戻る)

書式

SYSTEM

機能

QB環境下でSYSTEM文が実行されるとQB画面に戻ります．ただし，QB起動時に/runオプションが指定されていた場合は，SYSTEM文が実行されるとOSに戻ります．

.EXEファイルの中でSYSTEM文が実行されると，すべてのファイルをクローズしてOSに戻ります．

ダイレクトモードでSYSTEMコマンドを入力すると，QBを終了してOSに戻ります．

TAB　（表示位置の指定）　関数

書式

TAB(n)
　　　└画面左端からの桁位置

機能

PRINT, LPRINT 文と共に用い，スクリーンまたはプリンタの左端(桁位置は1)からの表示/印字位置をnに設定します.

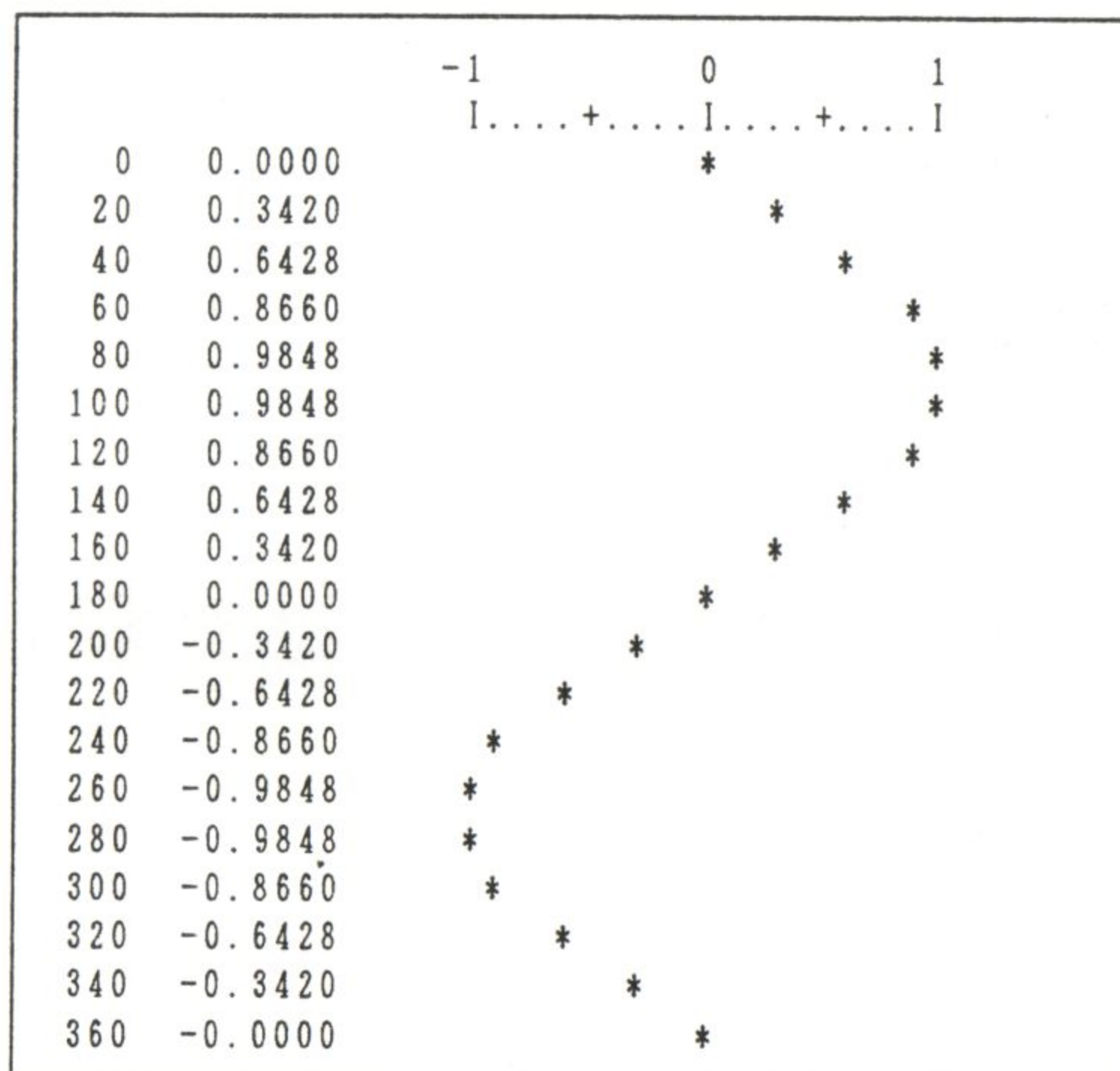
現在の表示/印字位置より小さいnの値を指定すると，1行改行されてn桁目に表示/印字されます.

nの値が1行の幅より大きい場合は，次の行に移り，(n MOD 幅)の位置に表示/印字されます.

nの値に1より小さい値を指定すると，1とみなされます.

例

```
' ---/* サイン・カーブ */---

rd = 3.14159 / 180
PRINT TAB(19); "-1         0         1"
PRINT TAB(20); "I....+....I....+....I"
FOR x = 0 TO 360 STEP 20
    ts = 30 + 10 * SIN(x * rd)  ←―――― 表示位置
    PRINT USING "##### ###.####"; x; SIN(x * rd);
    PRINT TAB(ts); "*"
NEXT x
```

```
                   -1         0         1
                    I....+....I....+....I
    0    0.0000               *
   20    0.3420                  *
   40    0.6428                     *
   60    0.8660                        *
   80    0.9848                         *
  100    0.9848                         *
  120    0.8660                        *
  140    0.6428                     *
  160    0.3420                  *
  180    0.0000               *
  200   -0.3420            *
  220   -0.6428         *
  240   -0.8660      *
  260   -0.9848     *
  280   -0.9848     *
  300   -0.8660      *
  320   -0.6428         *
  340   -0.3420            *
  360   -0.0000               *
```

TAN　（タンジェント）　関数

書式　TAN(x)

機能　x のタンジェントを求めます．x の単位はラジアン．

例　SIN 参照．

TIME$　（現在の時刻の取得）　関数

書式　TIME$

機能　現在の時刻を，hh：mm：ss の 8 文字の文字列で取得します．

hh　：　時(00～23)
mm　：　分(00～59)
ss　：　秒(00～59)

例

```
' ---/* 時刻の取得と設定 */---

CLS
PRINT "現在の時刻-->"; TIME$                                    ←――― 時刻の取得
INPUT "時,分,秒 "; hour$, minute$, second$
TIME$ = hour$ + ":" + minute$ + ":" + second$  ←―― 時刻の設定
```

```
現在の時刻-->15:32:16
時,分,秒 ? 16,10,00
```

TIME$　（時刻の設定）

書式　TIME$ = str
　　　　　　　↑時刻を示す文字列

機能　str で示す時刻に現在時刻を設定します．str は hh：mm：ss の文字列です．mm：ss または ss を省略すると，省略されたものは00に設定されます．

hh　：　時(00～23)
mm　：　分(00～59)
ss　：　秒(00～59)

TIMER (午前0時からの経過秒の取得) 関数

書 式

TIMER

機 能

午前0時からの経過秒を取得します.

例

```
st& = TIMER
PRINT "これから30秒間眠ります...
WHILE TIMER - st& < 30
WEND
```

…30秒間経過するまでWHILE～WEND ループをまわる.

TIMER ON/OFF/STOP (タイマイベント・トラッピングの開始/禁止/停止)

書 式

① TIMER ON
② TIMER OFF
③ TIMER STOP

機 能

①タイマイベント・トラッピングの開始
②タイマイベント・トラッピングの禁止
③タイマイベント・トラッピングの停止. 停止中に発生したトラッピングは記憶されており, トラッピングが許可(開始)されると, ただちにイベント処理ルーチンに分岐します.

例

ON event GOSUB 参照.

TRON/TROFF(トレース・オン/トレース・オフ)

書　式

① TRON
② TROFF

機　能

①トレースモードをオンにします．つまり，1行ごと実行しながら実行行を反転表示します．
②トレースモードをオフにします．

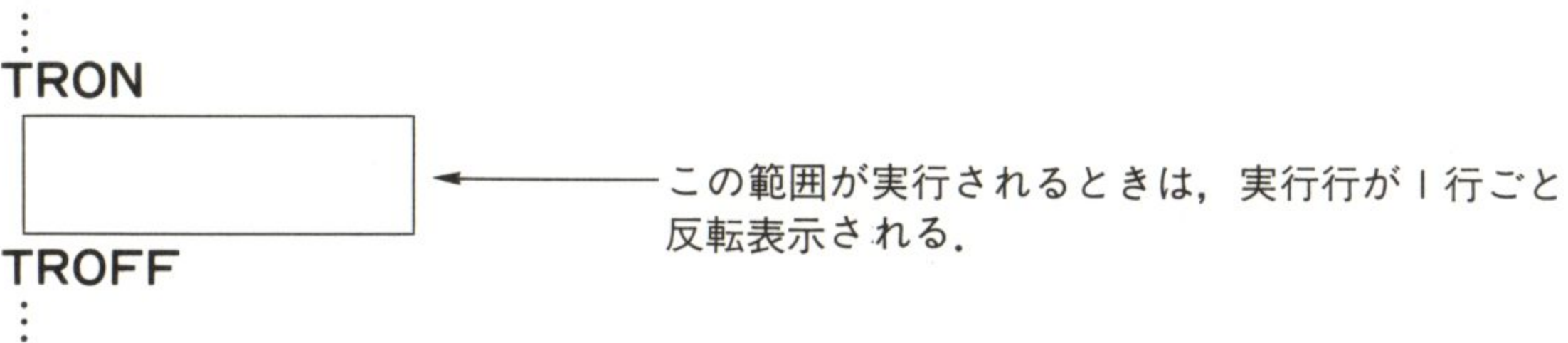
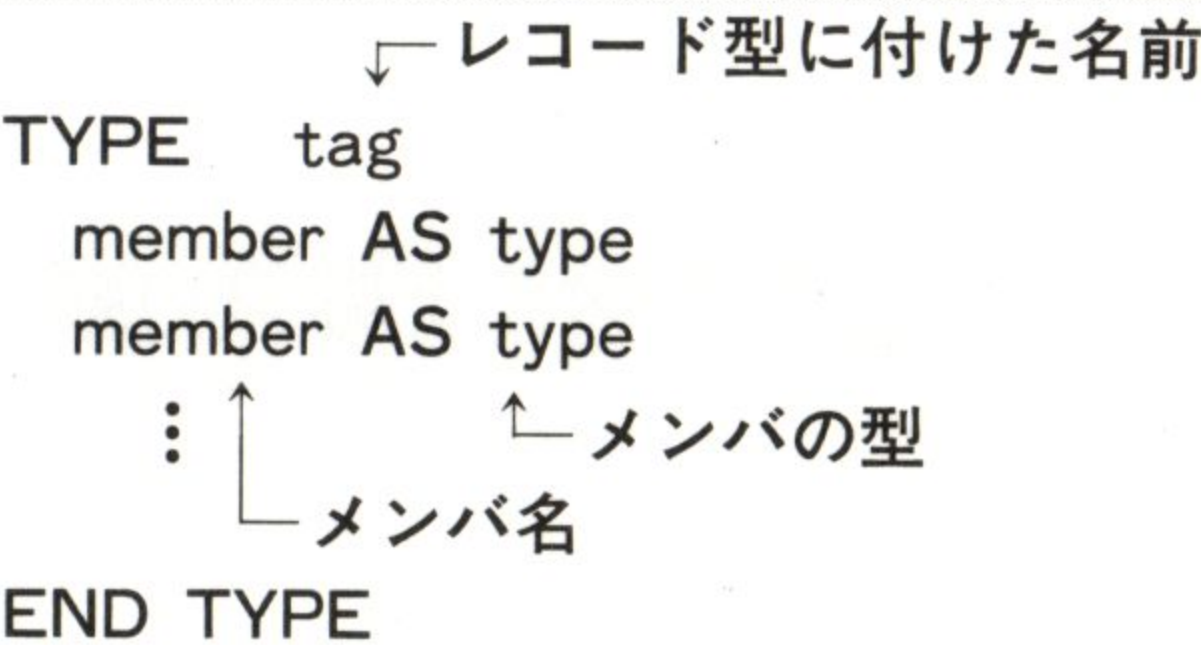
TRON は，D/デバッグメニューの T/トレースを指定したのと同じ効果が得られます．

例

```
:
TRON
┌──────────┐
│          │  ←── この範囲が実行されるときは，実行行が1行ごと
└──────────┘       反転表示される．
TROFF
:
```

TYPE　(レコード型の定義)

書　式

```
       ┌─レコード型に付けた名前
TYPE   tag
  member AS type
  member AS type
    : ↑         └─メンバの型
      └─メンバ名
END TYPE
```

機　能

tag という名のレコード型を定義し，そのレコードを構成するメンバのメンバ名とメンバの型を定義します．

メンバの型には，INTEGER，LONG，SINGLE，DOUBLE，固定長文字列型，ユーザ定義型(レコード型)を指定できます．可変長文字列はメンバの型としては使えないことに注意してください．

レコード型変数は，tag を用いて，DIM，REDIM，COMMON，STATIC，SHARED 文で行います．

レコードの各メンバの参照は，ピリオド(.)を用いて，

レコード型変数．メンバ名

で行います．

レコード型の定義はプロシージャの中では行えません．

例

```
' ---/* 木星の４大衛星 */---

TYPE Eisei                    ' Ｅｉｓｅｉ型の定義
    Star AS STRING * 10
    Kodo AS INTEGER
    Shuki AS SINGLE
END TYPE

DIM a(4) AS Eisei             ' Ｅｉｓｅｉ型変数の宣言
DIM b(4) AS Eisei

FOR k = 1 TO 4
    READ a(k).Star, a(k).Kodo, a(k).Shuki
NEXT k                          ↑――――――― メンバの参照

FOR k = 1 TO 4
    b(k) = a(k)               ' レコード・データの一括代入
    PRINT b(k).Star, b(k).Kodo, b(k).Shuki
NEXT k

DATA Io,5,1.7691,Europa,6,3.5512
DATA Ganymede,5,7.1545,Callisto,6,16.6890
```

```
Io              5           1.7691
Europa          6           3.5512
Ganymede        5           7.1545
Callisto        6           16.689
```

UBOUND （配列添字の上限値の取得） 関数

書式

UBOUND(array[,d])

- array ← 配列名
- d ← 添字の次元

機能

配列 array の d 次の添字の上限値を取得します.

LBOUND, UBOUND は引数渡しされた配列のサイズを取得するのに使います.

```
DIM A(-10 TO 20, 5 TO 10)

UBOUND(A, 1)    …  20を返す.
UBOUND(A, 2)    …  10を返す.
```

例

LBOUND 参照.

UCASE$　(大文字に変換)　関数

書式

UCASE$(str)
　　　　　↑文字列

機能

文字列中の小文字を大文字に変換して返します．対象となる文字は1バイト文字，2バイト文字のa～zです．

例

```
INPUT A$
IF UCASE$(A$)="YES"  THEN～
```

…A$にYes，yesなどが入力されてもYESとして比較できる．

UNLOCK (他のプロセスからのアクセス禁止を解除)

☞ LOCK参照．

VAL　(文字列を数値に変換)　関数

書式

VAL(str)
　　↑文字列

機能

文字列strを数値に変換します．先頭に&hまたは&Hがあれば16進数とみなされます．文字列の先頭にある空白，タブ，ラインフィードは読み飛ばされますが，文字列の途中にあってはいけません．もし，数字文字でない文字があると，その直前までの文字列を数値に変換します．最初の文字が数字文字でない場合は0を返します．

例

```
VAL("123")    … 123 を返す.
VAL("&HFF")   … 255 を返す.
VAL("FF")     … 0 を返す.
```

```
' ---/* 文字列を数値に変換 */---

FOR k = 1 TO 5
    READ a$
    PRINT VAL("&h" + a$)
NEXT k
DATA ff,1a,1b,80,7f
```

…　DATAに置かれている16進文字列データを数値に変換する．

VARPTR/VARSEG（変数のアドレスのオフセット値/セグメント値の取得）　関数

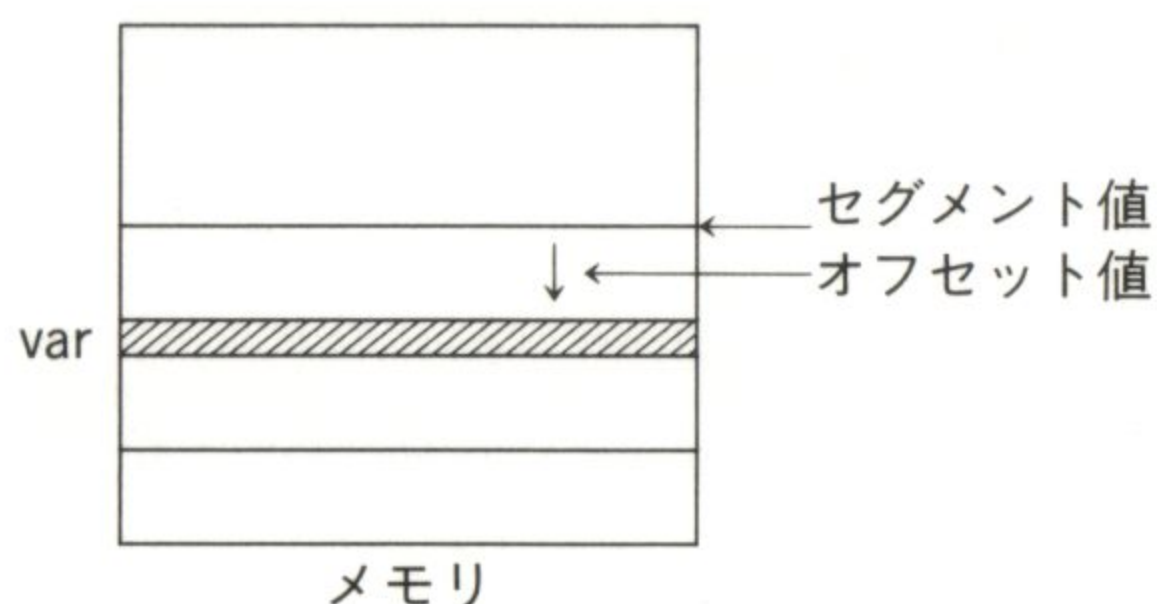
書式

VARPTR(var)
VARSEG(var)
　　　　　└─変数名

機能

変数 var が割り当てられているメモリ上のアドレスのオフセット値とセグメント値を取得します．

配列のアドレスを取得する場合は，var として配列の第 1 要素を指定してください．

var
セグメント値
オフセット値
メモリ

例

```
' ---/* 変数への直接アクセス */---

CONST N = 100
DIM a%(N)

segment = VARSEG(a%(0))
offset = VARPTR(a%(0))

DEF SEG = segment
FOR adr = offset TO offset + N * 2 STEP 2
    POKE adr, &H1
    POKE adr + 1, &H0
NEXT adr

FOR k = 0 TO N
    PRINT a%(k);
NEXT k
```

…配列の先頭アドレスを取得し，POKE を用いてそのアドレスを直接アクセスする．

DEF SEG = segment ← セグメントの設定
POKE adr, &H1 ← 下位バイト
POKE adr + 1, &H0 ← 上位バイト

VARPTR$　（変数のアドレスを文字列として取得）　関数

書式

```
VARPTR$(var)
        ↑─変数名
```

機能

変数 var が割り当てられているメモリ上のアドレスのオフセット値を文字列として取得します.

VARPTR$は, DRAW の引数に用いる変数のアドレスを取得するのに使われます.

例

DRAW 参照.

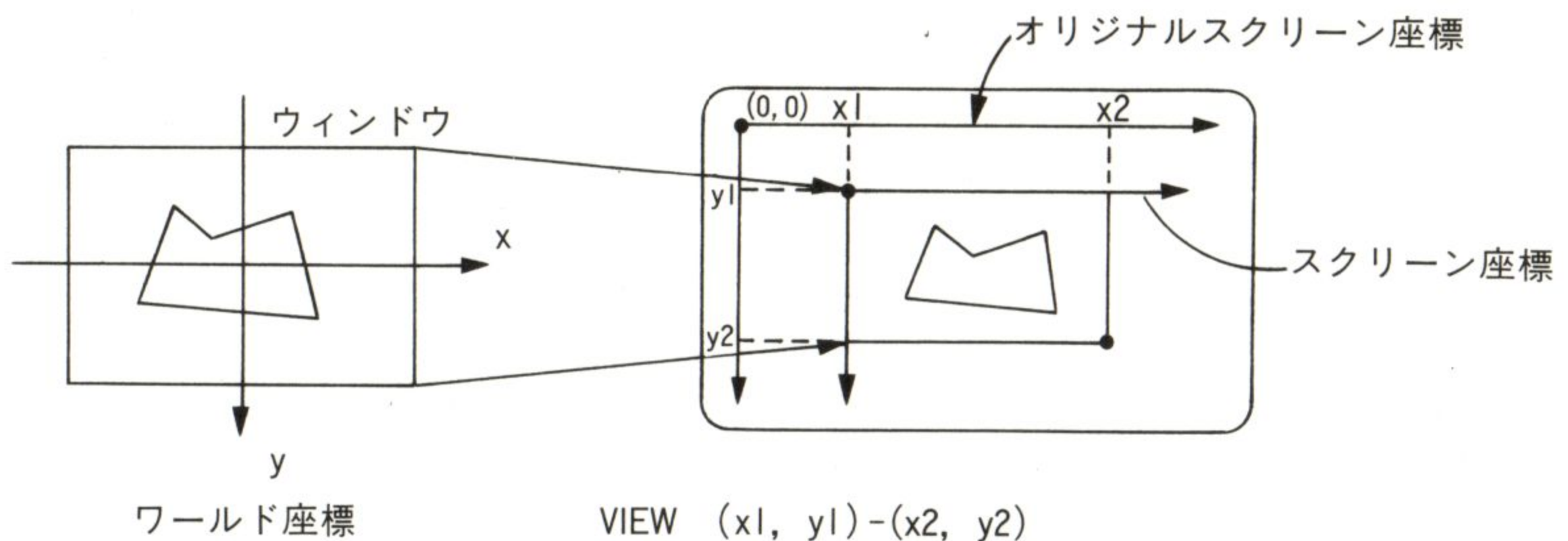
VIEW　（ビューポートの設定）

書式

```
VIEW [[SCREEN](x1, y1) - (x2, y2)[,[color][,border]]]
```

- (x1, y1) - (x2, y2)：ビューポートを示す範囲
- color：ビューポート内を塗る色
- border：ビューポートの境界線の色

機能

VIEW (x1, y1) - (x2, y2)はオリジナルスクリーン座標上の (x1, y1) - (x2, y2)をビューポートに設定し, その領域に WINDOW で指定された領域を表示します.

これに対し, VIEW SCREEN(x1, y1)-(x2, y2)は, WINDOW で指定された領域をオリジナルスクリーン座標上((0,0)-(639 , 399)) に表示しますが, 実際に表示する範囲を(x1, y1) - (x2, y2) の範囲に限定します.

オリジナルスクリーン座標

ウィンドウ

x

y

x1

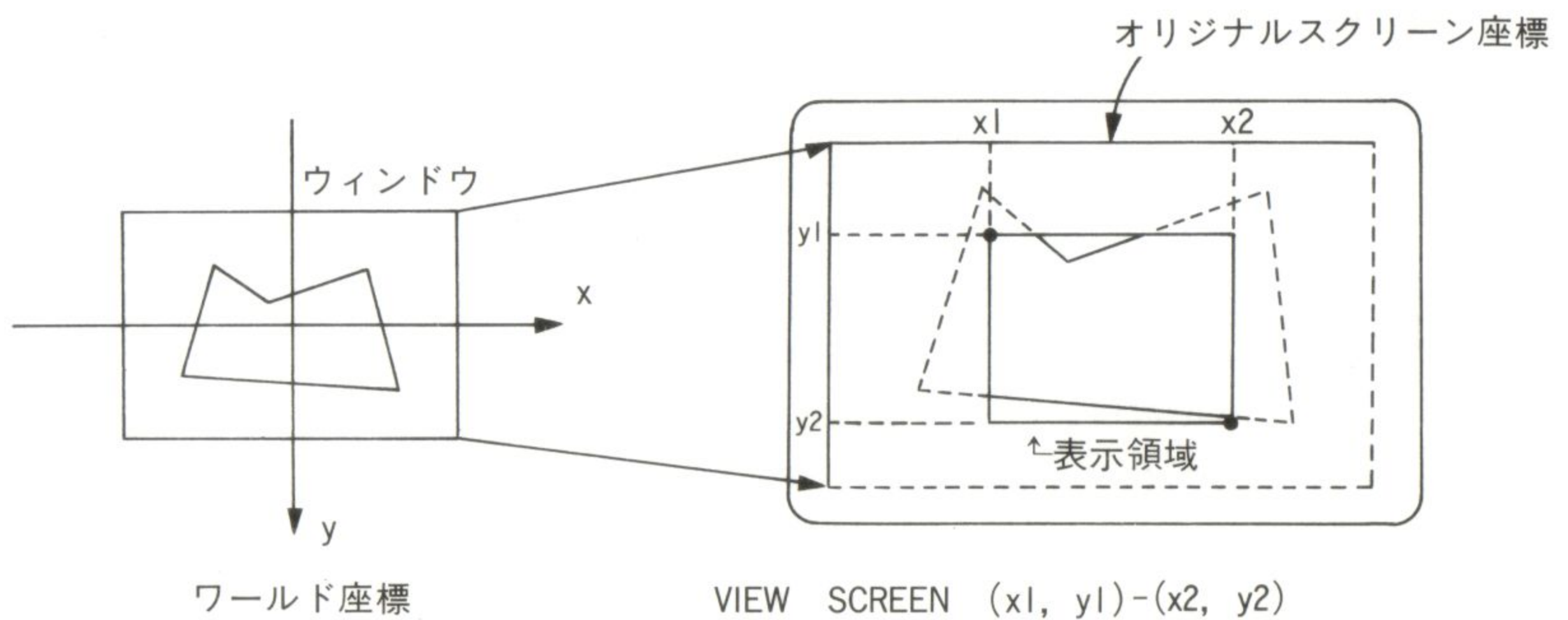
x2

y1

y2

表示領域

ワールド座標

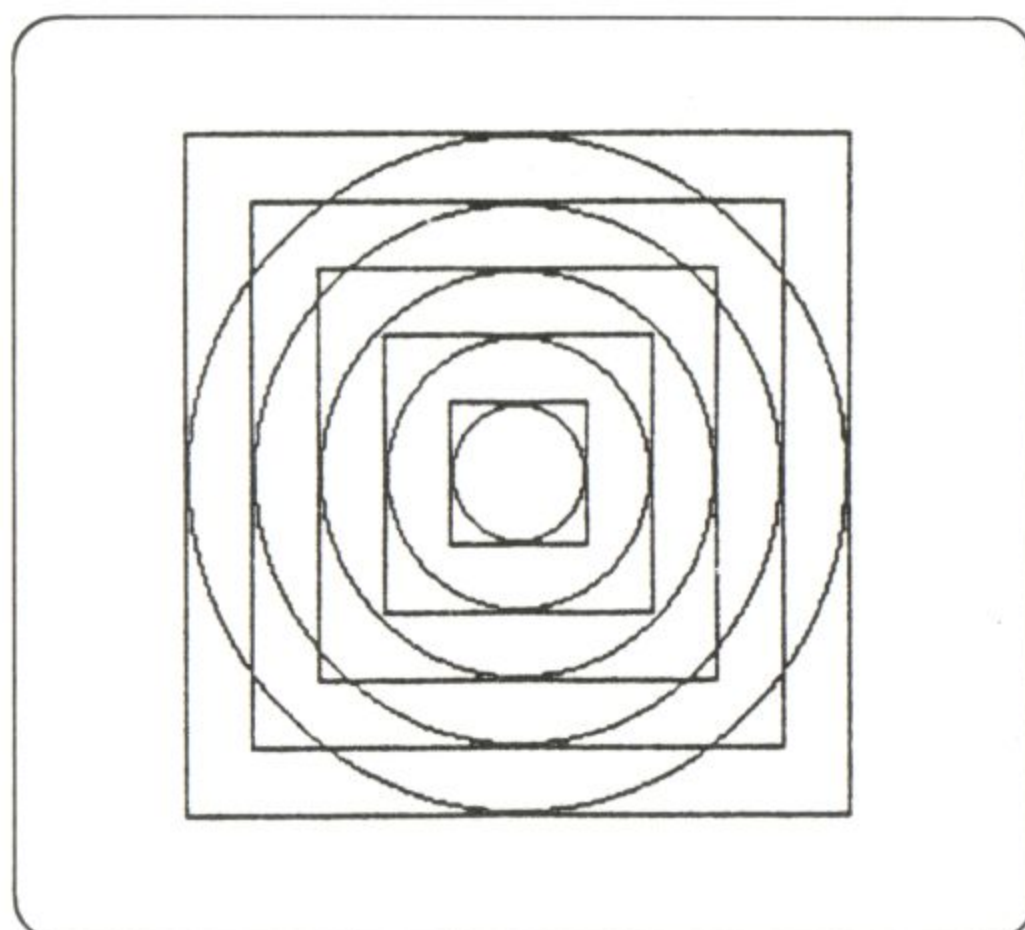
VIEW　SCREEN　(x1, y1)-(x2, y2)

color はビューポート内を塗る色，border はビューポートを示す四角形の枠を描く色です．

引数を省略し，単に VIEW とした場合は，画面全体がビューポートとなります．

例

```
' ---/* ビューポートの設定 */---

SCREEN 0: CLS
WINDOW (-50, -50)-(50, 50)

x1 = 1: y1 = 1
x2 = 201: y2 = 201

FOR k = 0 TO 4
    VIEW (x1, y1)-(x2, y2), , 1
    CIRCLE (0, 0), 50
    x1 = x1 + 20: y1 = y1 + 20
    x2 = x2 - 20: y2 = y2 - 20
NEXT k
```

ビューポート枠を青色で表示

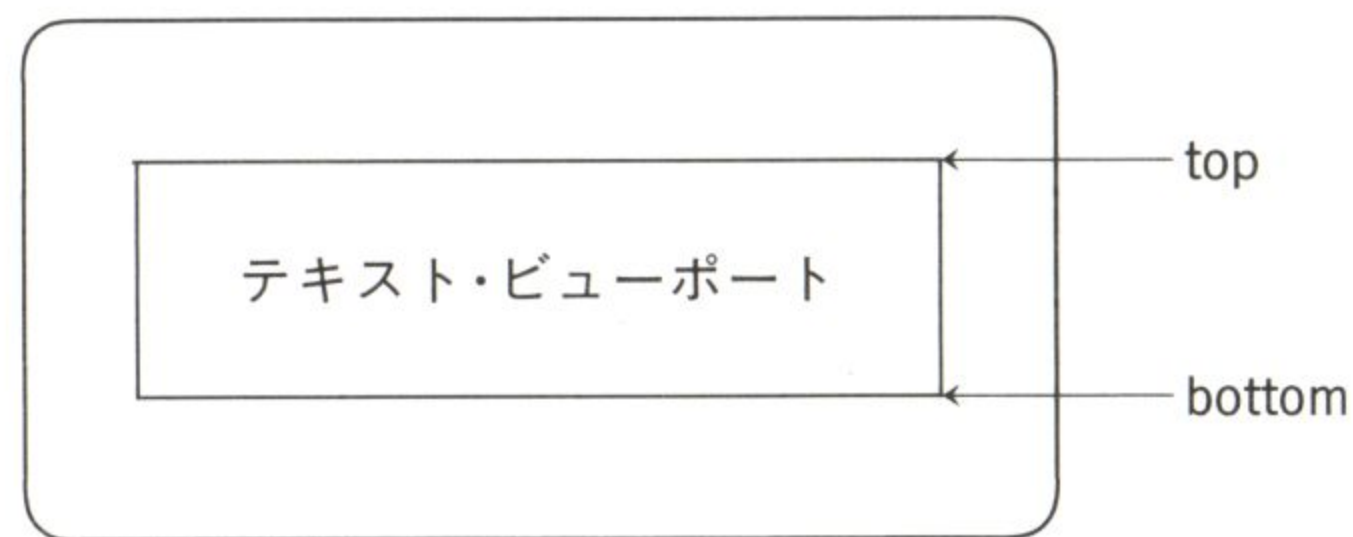
ビューポートを狭くしていく

VIEW PRINT （テキスト・ビューポートの上下行の指定）

書　式

```
VIEW PRINT [top TO bottom]
```

top：最上行
bottom：最下行

機　能

テキスト・ビューポートの上下行を指定します．

画面の最上行は1行からです．
引数を省略すると画面全体がテキスト・ビューポートになります．

例

```
' ---/* テキスト・ビューポートの設定 */---

VIEW PRINT 5 TO 15
FOR k = 1 TO 100
    PRINT k, "Hello"
NEXT k
```

VIEW PRINT 5 TO 15 ← テキスト・ビューポートを5行～15行に設定．この範囲が表示域となる．

WAIT （入力ポートの状態を調べるあいだプログラムを停止）

書　式

```
WAIT n, and-expression[, xor-expression]
```

n：ポート番号
and-expression：ANDを行う整数式
xor-expression：XORを行う整数式

機　能

ポートnからデータを読み，そのデータとxor-expressionとのXORをとり，さらにその値とand-expressionとのANDをとります．そしてその値が0なら再びポートからデータを読み同じことをくり返します．値が0でなくなったときにWAIT文の次の文に進みます．xor-expressionは省略することができ，そのときは，AND演算から行われます．

例

```
' ---/* プリンタ・ポートへの1文字出力 */---

DECLARE SUB lputc (c%)

lputc (ASC("a"))
lputc (ASC("b"))
lputc &HD: lputc &HA     ' 改行

SUB lputc (c%)
    WAIT &H42, 4 ◄────── ポート&H42のデータの第2ビットが
    OUT &H40, c%          1になるまで待つ.
    OUT &H46, 14
    OUT &H46, 15
END SUB
```

…lputc(c%)は ASCII コード c%に対応する文字をプリンタに出力する.

WHILE～WEND　（前判定反復）

書式

```
WHILE condition
  ⋮         ↑─条件式
WEND
```

機能

条件式 condition が真のあいだ，ループをくり返します．WHILE～WEND は DO WHILE～LOOP と同じです．ただし，DO～LOOP は DO EXIT によりループ外に脱出できますが，WHILE～WEND では GOTO で脱出するしかありません．

例

```
' ---/* 前判定反復 */---

Sum = 0

INPUT a
WHILE a <> -9999
    Sum = Sum + a
    INPUT a
WEND

PRINT "合計="; Sum
```

WIDTH　　（画面幅の設定）

書式

```
WIDTH [column][,line]
        ↑          ↑─行数
        └─桁数
```

機能

テキスト画面のサイズを設定します．Ver.4.2ではcolumnに80，lineに25以外の値を指定することはできません．Ver.4.5では，lineに20も指定できます．

WIDTH（ファイルおよびデバイスの1行の長さの設定）

書式

```
① WIDTH #n, width
          ↑     ↑─1行の長さ
          └─ファイル番号
② WIDTH device, width
        ↑         ↑─1行の長さ
        └─デバイス・ファイル名
③ WIDTH LPRINT width
               ↑─1行の長さ
```

機能

①ファイル番号nでオープンされているデバイス・ファイルの1行の長さをwidthに設定します．

②deviceで示されるデバイス（"cons:"など）の1行の長さをwidthに設定します．この命令が効果を現すのは，このデバイスが次のOPEN文でオープンされてからあとです．

③以後のLPRINT文を使用するラインプリンタの1行の長さをwidthに設定します．

例

```
' ---/* プリンタの出力幅の設定 */---

a$ = "abcdefghijklmnopqrstuvwxyz"

WIDTH LPRINT 5
LPRINT a$

WIDTH LPRINT 10
LPRINT a$
```

```
abcde
fghij
klmno
pqrst
uvwxy
z
abcdefghij
klmnopqrst
uvwxyz
```

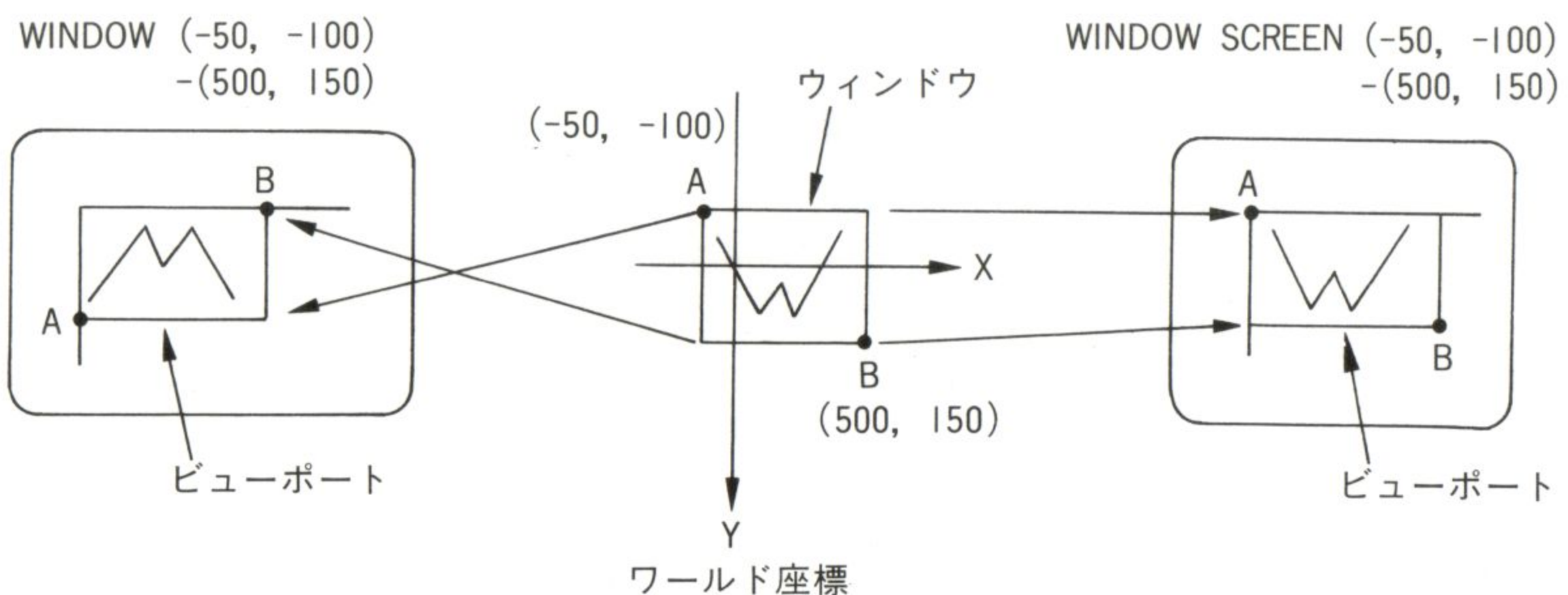
WINDOW　　(ウィンドウの設定)

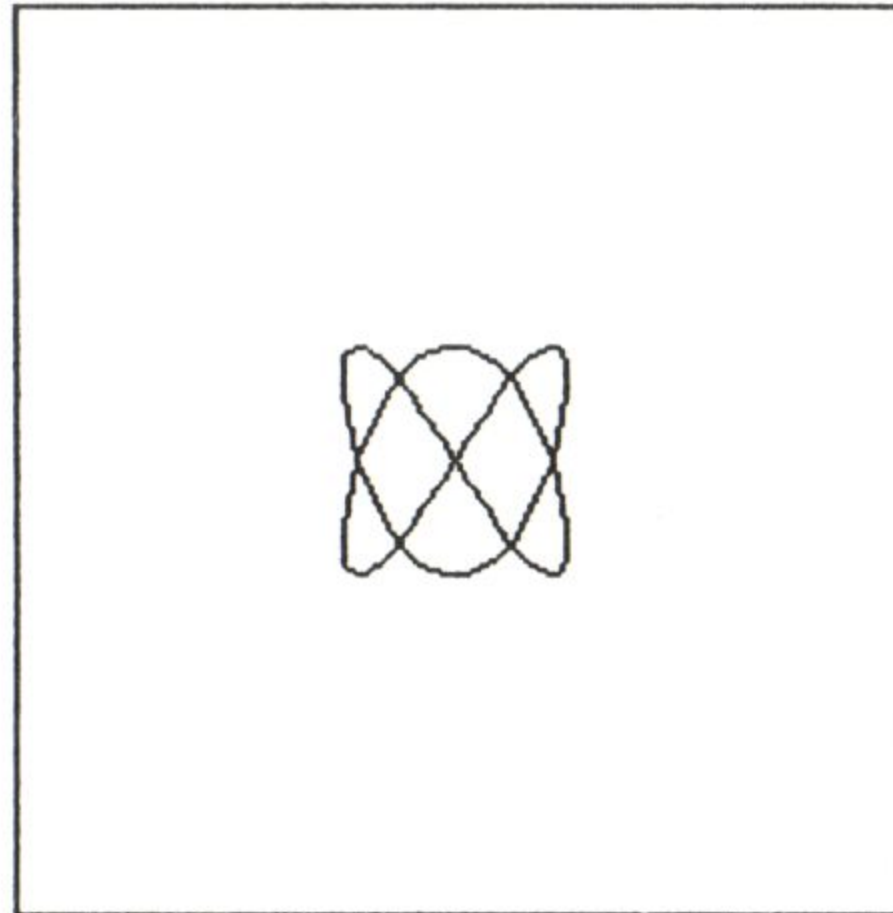
書式

```
WINDOW [[SCREEN] (x1, y1) - (x2, y2)]
                  └─ウィンドウの座標
```

機能

ワールド座標の(x1, y1) - (x2, y2)の範囲を，ウィンドウとして設定します．

Quick BASIC では，WINDOW 文によるウィンドウの切り方に2通りの方法があります．WINDOW 文に SCREEN オプションを付けると，下方を Y 座標の正の向きとするワールド座標をそのままの向きでビューポートに取り出し，SCREEN オプションを付けないと，Y 方向を逆転してビューポートに取り出します．

例

```
' ---/* ウィンドウの設定 */---

SCREEN 0: CLS

rd = 3.14159 / 180
x1 = -5: y1 = -5
x2 = 5: y2 = 5

VIEW (200, 100)-(400, 300), , 1 ←───ビューポートの設定
FOR k = 1 TO 10
    CLS
    WINDOW (x1, y1)-(x2, y2) ←───ウィンドウの設定
    FOR j = 0 TO 360 STEP 4
        x = SIN(2 * j * rd): y = SIN(3 * j * rd)
        IF x = 0 THEN
            PSET (x, y)
        ELSE                                  ←──リサジュー曲線の描画
            LINE -(x, y)
        END IF
    NEXT j
    x1 = x1 + .5: y1 = y1 + .5  ←──ウィンドウの範囲を狭めていく(図形は
    x2 = x2 - .5: y2 = y2 - .5      拡大されていく)
NEXT k
```

WRITE　　（スクリーンへの出力）

書式

```
WRITE arg, …
      ↑出力データ
```

機能

argで示される出力データをスクリーンに表示します．このとき，各データの区切りにカンマ(,)が挿入され，文字列の表示では，二重引用符(")で囲まれます．数値データを表示する際，前後に空白を付け加えません．

例

```
a = 10
b$ = "Hello"
c = 20

WRITE a, b$, c
PRINT a, b$, c
```

```
10,"Hello",20
 10            Hello            20
```

WRITE #　（シーケンシャル・ファイルへのライト）

書式

```
WRITE #n, arg, …
       ↑    ↑出力データ
       └ファイル番号
```

機能

argで示される出力データをファイル番号nのファイルにライトします．各出力データはカンマ(,)で区切ります．出力に際して各データの区切りにカンマ(,)が挿入され，文字列データは，二重引用符(")で囲まれて出力されます．この点がPRINT #と異なります．

例

```
' ---/* シーケンシャルファイルの書き込み */---

OPEN "man.dat" FOR OUTPUT AS #1

INPUT "Name, year"; Name$, Year

DO WHILE Year >= 0
    WRITE #1, Name$, Year
    INPUT "Name, Year"; Name$, Year
LOOP
CLOSE #1
```

```
A>dump man.dat

 Dump Version 3.00

00000000  22 41 6E 6E 22 2C 31 38-0D 0A 22 43 61 6E 64 79   "Ann",18.."Candy
00000010  22 2C 32 31 0D 0A 22 45-6C 75 7A 61 22 2C 31 39   ",21.."Eluza",19
00000020  0D 0A 22 52 6F 6C 6C 61-22 2C 32 30 0D 0A         .."Rolla",20..
```

メタコマンド

$INCLUDE　（ファイルのインクルード）

書式

{REM | ' }$INCLUDE: 'filename'
　　　　　　　　　　↑取り込むファイル名

機能

filenameで示されるファイルを，この位置にインクルード(取り込み)します．

インクルード・ファイルは，テキストファイルでなければなりません．インクルード・ファイルの作り方は，第2章2-4の7を参照してください．プログラムをインクルード・ファイルにする場合は，保存形式を「T/テキストファイル」に指定してください．

インクルード・ファイルには，プロシージャ(SUBやFUNCTION)を記述できません．

filenameのファイルタイプを省略すると，.BASとみなされます．

例

```
'$INCLUDE:'b:my.bi'
DIM a AS Eisei
PRINT Lpx, Lpy
Gcopy
```

インクルード

```
' -------------------------------
'   インクルード・ファイルの例
' -------------------------------

TYPE Eisei
    Namae AS STRING * 10
    Kodo AS INTEGER
    Shuki AS SINGLE
END TYPE

COMMON SHARED Lpx, Lpy

DECLARE SUB Gcopy()
```

MY. BI

$DYNAMIC/$STATIC
(動的配列／静的配列の指定)

書　式

①{REM ｜' }$DYNAMIC
②{REM ｜' }$STATIC

機　能

①以後の配列を動的配列として割り当てます.
②以後の配列を静的配列として割り当てます.

$DYNAMIC/$STATIC を用いて指定しない場合，定数の添字を用いて宣言した配列は静的配列，変数の添字を用いて宣言した配列または最初に COMMON ステートメントで宣言されている配列は動的配列となります.

例

```
'$DYNAMIC
 DIM a(100)        ←動的配列

'$STATIC
 INPUT n
 DIM b(n)          ←変数添字なので'$STATIC が指定されても動的配列
 DIM c(100)        ←静的配列
```

第7章

Ver.4.2から Ver.4.5への変更点

7-1 Ver.4.5における主な変更点

Ver.4.2から Ver.4.5へのバージョンアップに伴い，変更された主な点は以下のとおりです．

- PC-9801の最初期型(もっとも古いタイプ)は Ver.4.5ではサポートしない．
- エミュレータモードで動作するインタプリタ QBE.EXE をサポート．
- 強力なオンラインヘルプ(QB アドバイザ)のサポート．
- PLAY，SOUND，UEVENT，SLEEP ステートメントの追加．
- SCREEN，COLOR，WIDTH ステートメント，SCREEN，CSNG$関数の機能変更．
- 環境変数 QBGRPNEC の導入．
- CIRCLE，LOCATE，PRINT ステートメントの高速化．
- [STOP]，[HELP] キーによるプログラム割り込みのサポート．
- 統合環境のエディタのスクロールスピードの向上と，カーソル惰性の改善．
- QB チュートリアルという学習ソフトのサポート．
- ヘルプ作成ユーティリティ HELPMAKE のサポート．
- ユーザライブラリとして汎用ライブラリ，グラフィックライブラリ，マウスライブラリをソースとライブラリでサポート．

7-2 Ver.4.5で提供されるソフトウェア

Ver.4.5では，セットアップディスク，プログラムディスク，ライブラリディスク，アドバイザディスクの4枚のフロッピィディスクが提供されます．以下のリストの中でコメントの付いているファイルが，Ver.4.5で追加された主なものです．

●セットアップディスク

```
 ドライブ A: のディスクのボリュームラベルは QB45_SETUP
 ディレクトリは A:¥

LEARN        <DIR>      89-11-15    4:50
SAMPLE       <DIR>      89-11-15    4:50
PACKING  LST      6974  89-11-15    4:50
QBE      DOC      4475  89-11-15    4:50
README   DOC      5532  89-11-15    4:50
SAMPLE   DOC      2942  89-11-15    4:50
SETUP    @@@      4099  89-11-15    4:50
SETUP    EXE     62193  89-11-15    4:50
        8 個のファイルがあります.
   614400 バイトが使用可能です.
```

```
 ドライブ A: のディスクのボリュームラベルは QB45_SETUP
 ディレクトリは A:¥LEARN

.            <DIR>      89-11-15    4:50
..           <DIR>      89-11-15    4:50
BALL     BAS      1306  89-11-15    4:50
HELP     HLP      4302  89-11-15    4:50
LEARN    EXE    167100  89-11-15    4:50  ←― QB チュートリアル
MENU     ATB      4103  89-11-15    4:50
MENU     TXT      4103  89-11-15    4:50
PRINT    HLP      1510  89-11-15    4:50
        8 個のファイルがあります.
   614400 バイトが使用可能です.
```

```
 ドライブ A: のディスクのボリュームラベルは QB45_SETUP
 ディレクトリは A:¥SAMPLE

.            <DIR>      89-11-15    4:50
..           <DIR>      89-11-15    4:50
ADVR_EX      <DIR>      89-11-15    4:50
BOOK         <DIR>      89-11-15    4:50
EXAMPLE      <DIR>      89-11-15    4:50
DEMO1    BAS      1934  89-11-15    4:50
DEMO2    BAS      2059  89-11-15    4:50
DEMO3    BAS      2140  89-11-15    4:50
QB       BI       1783  89-11-15    4:50
QCARDS   BAS     44644  89-11-15    4:50
QCARDS   DAT      2192  89-11-15    4:50
REMLINE  BAS     11712  89-11-15    4:50
SORTDEMO BAS     20536  89-11-15    4:50
TORUS    BAS     24275  89-11-15    4:50
WALTZ    BAS      4254  89-11-15    4:50
       15 個のファイルがあります.
   614400 バイトが使用可能です.
```

●プログラムディスク

```
 ドライブ A: のディスクのボリュームラベルは QB45_PGM
 ディレクトリは A:¥

BIN          <DIR>     89-11-15     4:50
MOUSE        <DIR>     89-11-15     4:50

        2 個のファイルがあります.
   152576 バイトが使用可能です.
```

```
 ドライブ A: のディスクのボリュームラベルは QB45_PGM
 ディレクトリは A:¥BIN

.                <DIR>     89-11-15     4:50
..               <DIR>     89-11-15     4:50
BC         EXE    111479   89-11-15     4:50
BRUN45A    EXE     92329   89-11-15     4:50
LIB        EXE     35643   88-07-26    10:52
LINK       EXE     69133   88-09-07    16:27
QB         EXE    311758   89-11-15     4:50  ←── 27KB 大きくなった
QB         INI        48   89-11-15     4:50
EM               <DIR>     89-11-10     4:50
        9 個のファイルがあります.
   152576 バイトが使用可能です.
```

```
 ドライブ A: のディスクのボリュームラベルは QB45_PGM
 ディレクトリは A:¥BIN¥EM

.                <DIR>     89-11-10     4:50
..               <DIR>     89-11-10     4:50
BRUN45E    EXE     93113   89-11-15     4:50
QBE        EXE    311870   89-11-15     4:50 ← エミュレータ版インタプリタ
        4 個のファイルがあります.
   152576 バイトが使用可能です.
```

```
 ドライブ A: のディスクのボリュームラベルは QB45_PGM
 ディレクトリは A:¥MOUSE

.                <DIR>     89-11-15     4:50
..               <DIR>     89-11-15     4:50
MOUSE      COM      3431   89-10-18     2:00
MOUSE      DOC     47900   89-10-18     2:00
MOUSE      SYS      8127   89-10-18     2:00
        5 個のファイルがあります.
   152576 バイトが使用可能です.
```

●ライブラリディスク

```
ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:¥

LIB          <DIR>      89-11-15    4:50
USERLIB      <DIR>      89-11-15    4:50
        2 個のファイルがあります.
   544768 バイトが使用可能です.
```

```
ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:¥LIB

.              <DIR>      89-11-15    4:50
..             <DIR>      89-11-15    4:50
EM             <DIR>      89-11-15    4:50
ABSOLUTE ASM       4520   89-12-15    4:20
BCOM45A  LIB     239343   89-11-15    4:50
BQLB45   LIB      25301   89-11-15    4:50
BRUN45A  LIB      25769   89-11-15    4:50
INTRPT   ASM      15099   89-12-15    4:20
QB       LIB       2075   89-11-15    4:50
QB       QLB       5814   89-11-15    4:50
       10 個のファイルがあります.
   544768 バイトが使用可能です.
```

```
ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:¥LIB¥EM

.              <DIR>      89-11-15    4:50
..             <DIR>      89-11-15    4:50
BCOM45E  LIB     229733   89-11-15    4:50
BRUN45E  LIB      25769   89-11-15    4:50
        4 個のファイルがあります.
   544768 バイトが使用可能です.
```

```
ドライブ A: のディスクのボリュームラベルは QB45_LIB
ディレクトリは A:¥USERLIB

.              <DIR>      89-11-15    4:50
..             <DIR>      89-11-15    4:50
ALL_MAKE BAT        544   89-11-15    4:50
DOSCALL  OBJ        435   89-11-15    4:50
GEN      BAS      16273   89-11-15    4:50 ┐
GEN      BI         927   89-11-15    4:50 │← 汎用ライブラリ
GEN      LIB      13871   89-11-15    4:50 │
GEN      QLB      13843   89-11-15    4:50 ┘
GENDEMO1 BAS       1094   89-11-15    4:50
GENDEMO2 BAS       4039   89-11-15    4:50
GEN_MAKE BAT        351   89-11-15    4:50
GRAPH    BAS       6906   89-11-15    4:50 ┐
GRAPH    BI         244   89-11-15    4:50 │← グラフィックライブラリ
GRAPH    LIB       6181   89-11-15    4:50 │
GRAPH    QLB       9295   89-11-15    4:50 ┘
GRPDEMO1 BAS        592   89-11-15    4:50
GRPDEMO2 BAS       2518   89-11-15    4:50
GRP_MAKE BAT        261   89-11-15    4:50
MOUSE    BAS      11059   89-11-15    4:50 ┐
MOUSE    BI         576   89-11-15    4:50 │← マウスライブラリ
MOUSE    LIB       5675   89-11-15    4:50 │
MOUSE    QLB       8083   89-11-15    4:50 ┘
```

```
MOUSEVNT ASM       2237  89-11-15    4:50
MOUSEVNT OBJ        225  89-11-15    4:50
MUSDEMO1 BAS       3392  89-11-15    4:50
MUSDEMO2 BAS       1385  89-11-15    4:50
MUS_MAKE BAT        281  89-11-15    4:50
       27 個のファイルがあります.
   544768 バイトが使用可能です.
```

●アドバイザディスク

```
ドライブ A: のディスクのボリュームラベルは QB45_ADVR
ディレクトリは A:¥

QBADVR       <DIR>      89-11-15    4:50
        1 個のファイルがあります.
   549888 バイトが使用可能です.
```

```
ドライブ A: のディスクのボリュームラベルは QB45_ADVR
ディレクトリは A:¥QBADVR

.            <DIR>      89-11-15    4:50
..           <DIR>      89-11-15    4:50
HELPMAKE EXE      47951  89-10-18    2:00 ←── ヘルプ作成ユーティリティ
QB45ADVR HLP     421739  89-11-15    4:50 ┐
QB45ENER HLP      69859  89-11-15    4:50 │
QB45QCK  HLP     121969  89-11-15    4:50 ├── QB アドバイザ
QB45USER HLP      14216  89-11-15    4:50 │
QB45USER QH       21450  89-11-15    4:50 ┘
        8 個のファイルがあります.
   549888 バイトが使用可能です.
```

7−3 セットアップ

Ver.4.2と同様にSETUPユーティリティを用いて，フロッピィまたはハードディスクにQBシステムをセットアップできます．

■ハードディスク

```
    ├── COMMAND.COM  ┐ SETUP ではコピーされないので必要に応じ
    ├── PRINT.SYS    │ ユーザが COPY する.
    ├── ATOK.DIC     │
    ├── ATOK6A.SYS   │
    ├── ATOK6B.SYS   ┘
    │            ├── NEW-VARS.BAT
    │            ├── NEW-CONF.SYS
    │            │
    │            ├── BIN ──┬── QB.EXE
    │            │         ├── BRUN45A.EXE
    │            │         ├── QB.INI
    │            │         ├── LINK.EXE
    │            │         ├── LIB.EXE
    │            │         ├── BC.EXE
    │            │         ├── HELPMAKE.EXE
    │            │         └── MOUSE.COM
    │            │
¥ ──┴── QB45 ────┼── LIB ──┬── BQLB45.LIB
                 │         ├── QB.LIB
                 │         ├── QB.QLB
                 │         ├── BRUN45A.LIB
                 │         ├── BCOM45A.LIB
                 │         ├── GEN.LIB
                 │         ├── GEN.QLB
                 │         ├── GRAPH.LIB
                 │         ├── GRAPH.QLB
                 │         ├── MOUSE.LIB
                 │         └── MOUSE.QLB
                 │
```

```
│                  ┌── GEN.BI
│                  ├── GRAPH.BI
├──INCLUDE─────────┼── MOUSE.BI
│                  ├── QB.BI
│                  ├── GNCONST.BI
│                  └── HWCONST.BI
│                  ┌── QB45QCK.HLP
├── ADVR ──────────┼── QB45ADVR.HLP
│                  ├── QB45ENER.HLP
│                  └── QB45USER.HLP
│    カレント
└──SOURCE      ……BASIC ソーステキストを置く.
```

SETUP により作成された NEW-VARS.BAT, NEW-CONF.SYS をもとに, 上記構成を実現する AUTOEXEC.BAT と CONFIG.SYS を作ります.

NEW-VARS.BAT

```
A>type new-vars.bat
PATH=A:¥QB45¥BIN
SET LIB=A:¥QB45¥LIB
SET INCLUDE=A:¥QB45¥INCLUDE
SET TMP=A:¥
MOUSE
```

↓

AUTOEXEC.BAT

```
A>TYPE ¥AUTOEXEC.BAT
PATH=A:¥QB45¥BIN
SET LIB=A:¥QB45¥LIB
SET INCLUDE=A:¥QB45¥INCLUDE
SET TMP=A:¥
CD A:¥QB45¥SOURCE
MOUSE
```

NEW-CONF.SYS

```
A>type new-conf.sys
SHELL=A:¥COMMAND.COM A:¥ /P
FILES= 20
BUFFERS=10
```

↓

CONFIG.SYS

```
A>TYPE ¥CONFIG.SYS
SHELL=A:¥COMMAND.COM A:¥ /P
FILES=20
BUFFERS = 10
DEVICE = ATOK6A.SYS
DEVICE = ATOK6B.SYS
DEVICE = PRINT.SYS
```

■1MBフロッピィ（SETUPを用いた場合）

SETUPを用いてフロッピィにセットアップすると，次のような3枚のフロッピィに分けられてしまい使い勝手はよくありません．

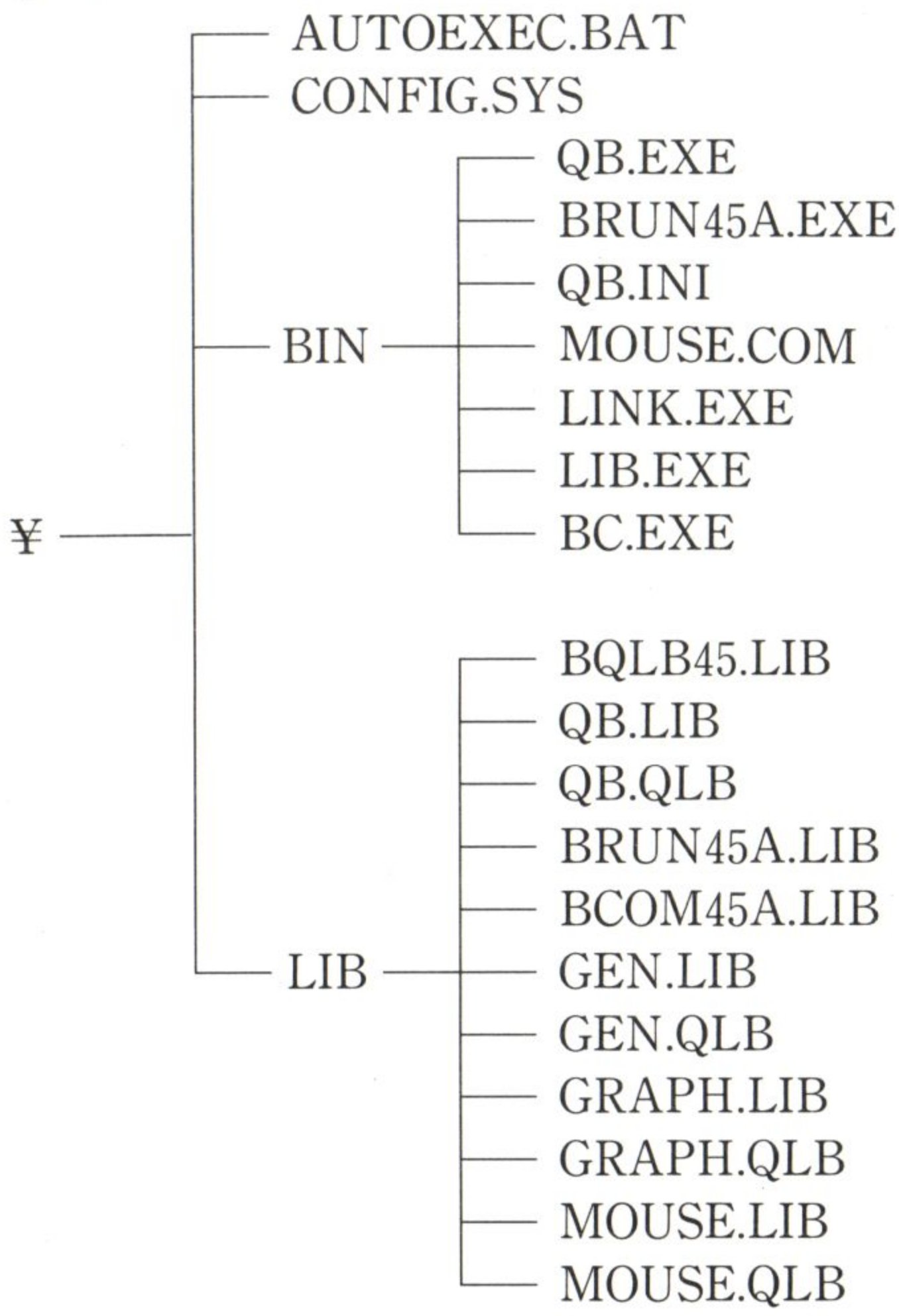
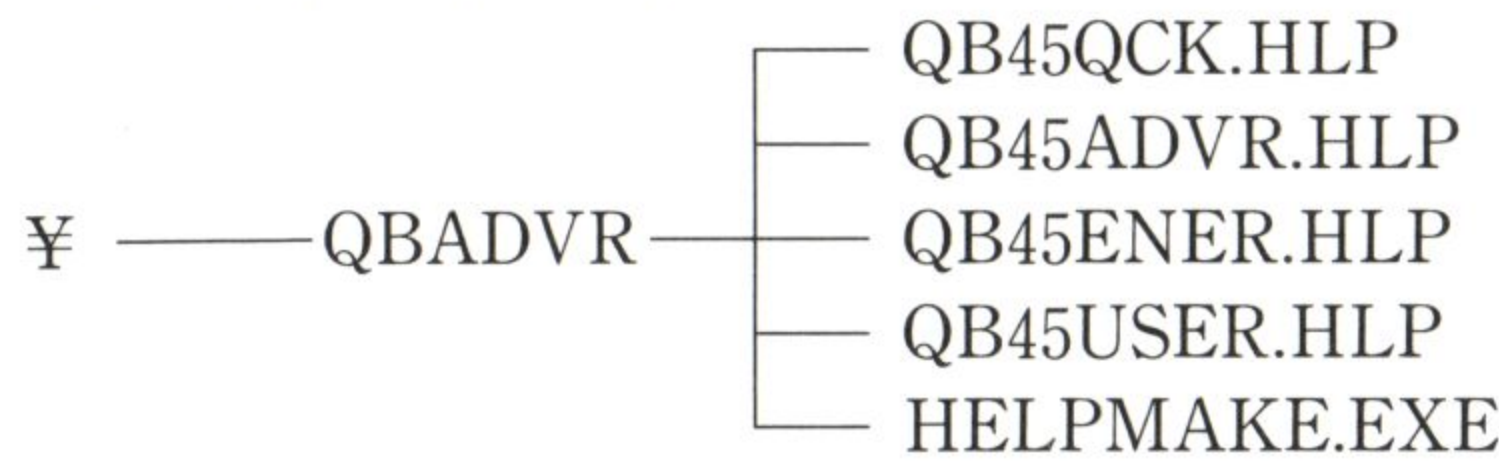
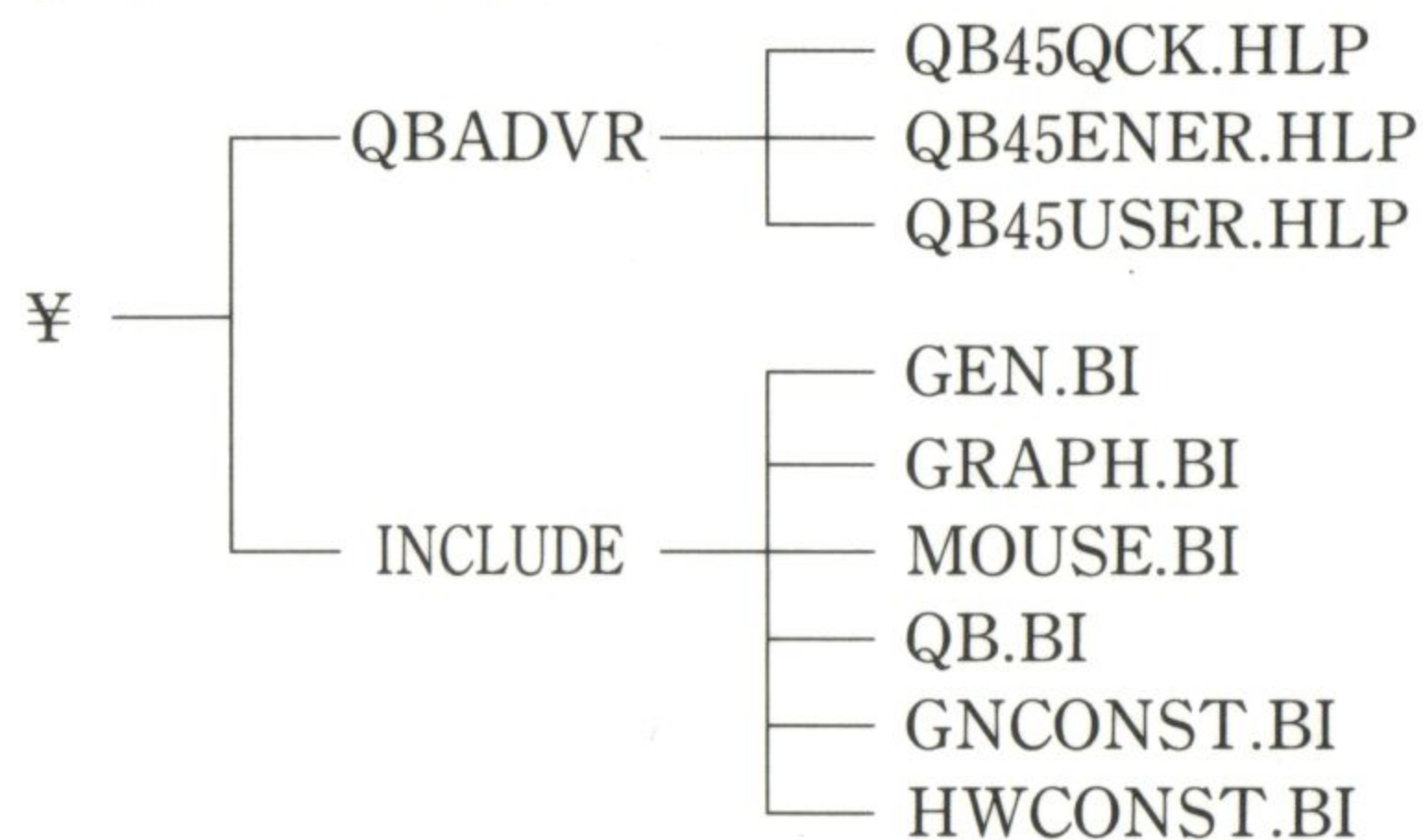
● QB起動ディスク

```
               ┌── AUTOEXEC.BAT
               ├── CONFIG.SYS
               │              ┌── QB.EXE
               │              ├── BRUN45A.EXE
               │              ├── QB.INI
               ├── BIN ───────┼── MOUSE.COM
               │              ├── LINK.EXE
               │              ├── LIB.EXE
¥ ─────────────┤              └── BC.EXE
               │
               │              ┌── BQLB45.LIB
               │              ├── QB.LIB
               │              ├── QB.QLB
               │              ├── BRUN45A.LIB
               │              ├── BCOM45A.LIB
               └── LIB ───────┼── GEN.LIB
                              ├── GEN.QLB
                              ├── GRAPH.LIB
                              ├── GRAPH.QLB
                              ├── MOUSE.LIB
                              └── MOUSE.QLB
```

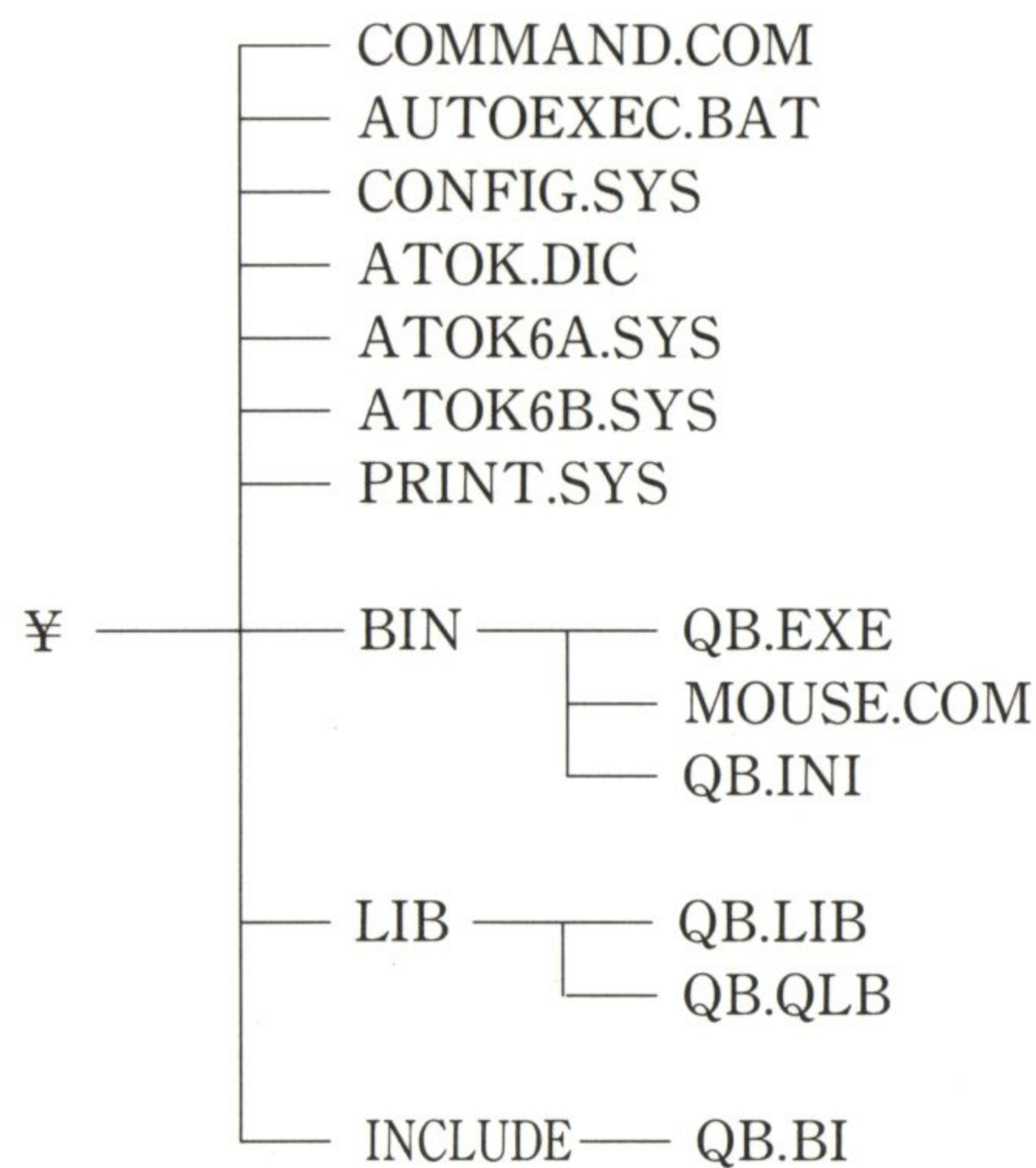
● QBアドバイザディスク

```
                    ┌── QB45QCK.HLP
                    ├── QB45ADVR.HLP
¥ ──── QBADVR ──────┼── QB45ENER.HLP
                    ├── QB45USER.HLP
                    └── HELPMAKE.EXE
```

● QB ワークディスク

```
                  ┌─ QB45QCK.HLP
     ┌─QBADVR ────┼─ QB45ENER.HLP
     │            └─ QB45USER.HLP
¥ ───┤
     │            ┌─ GEN.BI
     │            ├─ GRAPH.BI
     └─INCLUDE ───┼─ MOUSE.BI
                  ├─ QB.BI
                  ├─ GNCONST.BI
                  └─ HWCONST.BI
```

■1MB フロッピィ （独自にコピーした場合）

次のようにQBインタプリタだけのシステムにすれば，1枚のフロッピィに日本語辞書もいっしょに載せることができます．ただし，QBアドバイザは1枚に載せることはできません．

```
      ┌─ COMMAND.COM
      ├─ AUTOEXEC.BAT
      ├─ CONFIG.SYS
      ├─ ATOK.DIC
      ├─ ATOK6A.SYS
      ├─ ATOK6B.SYS
      ├─ PRINT.SYS
      │
¥ ────┼─ BIN ─────┬─ QB.EXE
      │           ├─ MOUSE.COM
      │           └─ QB.INI
      │
      ├─ LIB ─────┬─ QB.LIB
      │           └─ QB.QLB
      │
      └─ INCLUDE ─── QB.BI
```

7-4 メニュー画面

Ver.4.5では，QB起動直後のメニュー画面はEasyメニューになっています．Easyメニューは，Fullメニューのいくつかの機能を省略した初心者向けのメニューです．

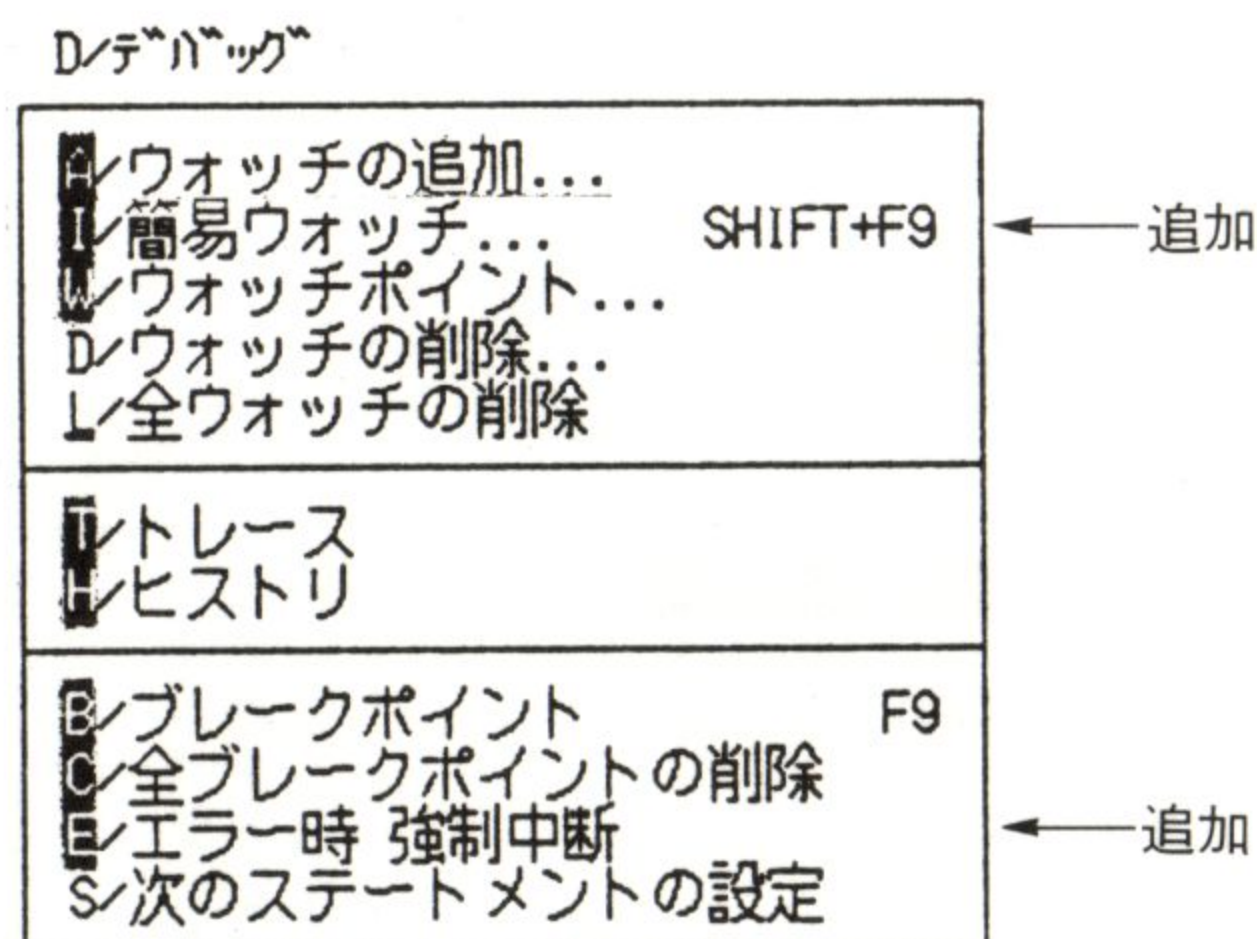
Easyメニューと Fullメニューの切り換えは，[O/オプション]—[F/Fullメニュー]により行います．

Ver.4.2と Ver.4.5のメニュー画面は，基本的には変わりませんが，以下のメニューで相違があります．

■ D/デバッグ

[I/簡易ウォッチ]と[E/エラー時強制中断]が追加されました．

D/デバッグ

```
A/ウォッチの追加...
I/簡易ウォッチ...          SHIFT+F9   ←追加
W/ウォッチポイント...
D/ウォッチの削除...
L/全ウォッチの削除
-----------------------------
T/トレース
H/ヒストリ
-----------------------------
B/ブレークポイント          F9
C/全ブレークポイントの削除
E/エラー時 強制中断                   ←追加
S/次のステートメントの設定
```

● I/簡易ウォッチ

プログラムを途中で中断した際，そのときの変数の値や式の状態(Trueなら－1，Falseなら0と表示)をダイアログボックスに表示します．ダイレクト・モードで変数の値を表示させるのと似た機能です．

```
式の値を表示します

─────────── 式 ───────────
sum ←変数または式

─────────── 値 ───────────
100 ←値または状態
```

● E/エラー時強制中断

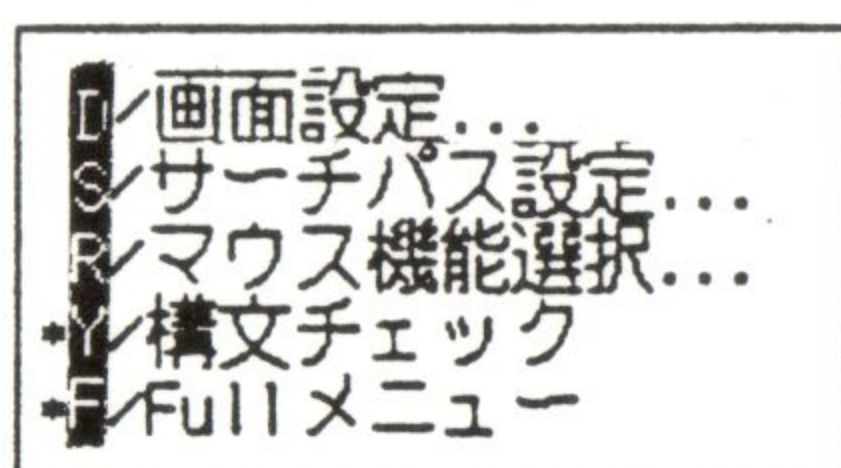
このサブメニューがONになっていると，ON ERROR文で指定されているエラー・ハンドラの先頭にブレークポイントを設定します．

つまり，エラーが発生すると，エラー・ハンドラの先頭でブレークすることになります．

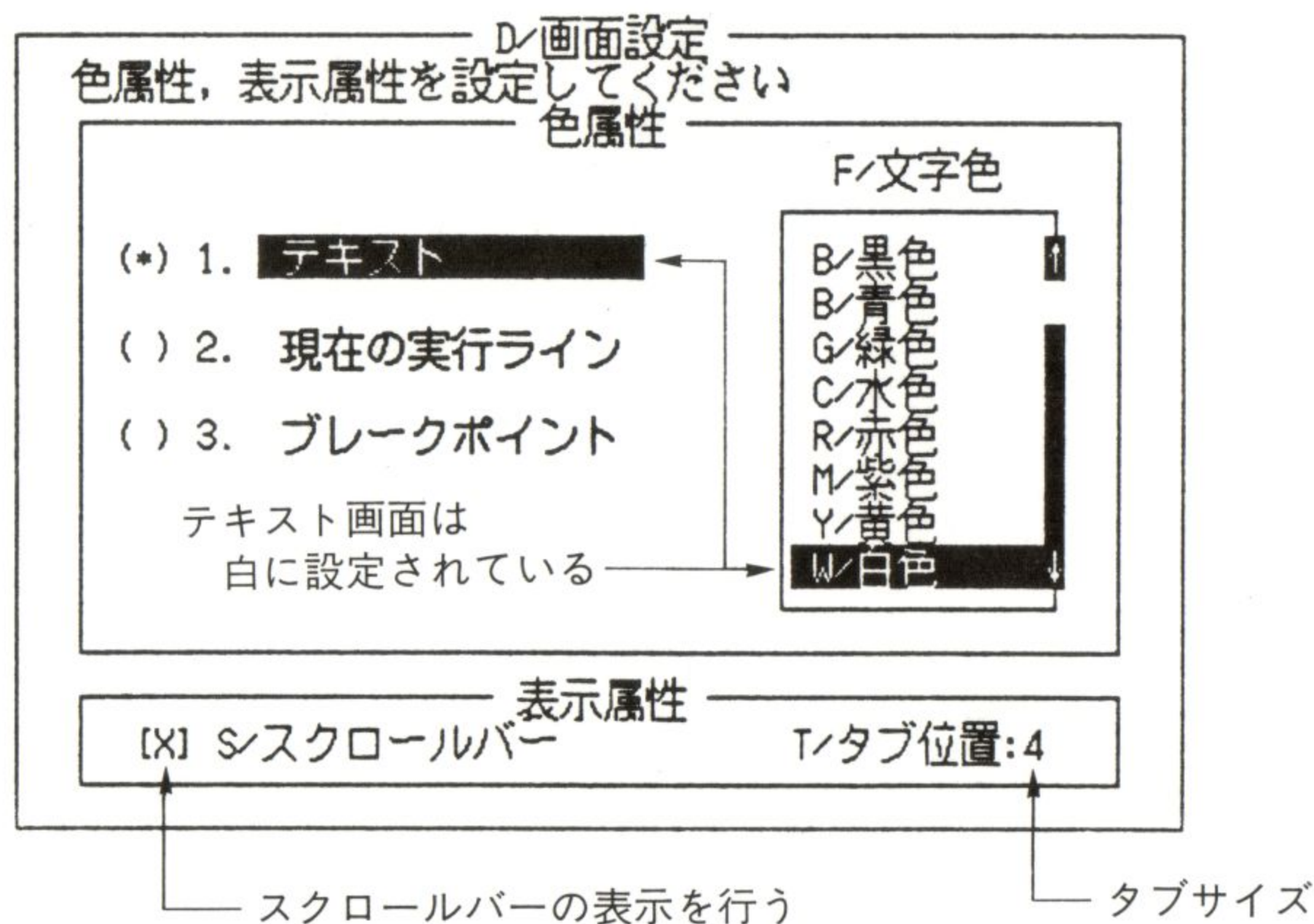
■ O/オプション

Ver.4.2では[V/表示]の中のサブメニューでしたが，Ver.4.5では独立したメニューとなり項目も追加されました．

O/オプション

D/画面設定...
S/サーチパス設定...
R/マウス機能選択...
*Y/構文チェック
*F/Fullメニュー

● D/画面設定...

テキスト，現在の実行ライン，ブレークポイントの色，スクロールバーの表示／非表示，タブサイズについての設定を行います．

D/画面設定
色属性，表示属性を設定してください
色属性
(*) 1. テキスト
( ) 2. 現在の実行ライン
( ) 3. ブレークポイント
テキスト画面は
白に設定されている
F/文字色
B/黒色
B/青色
G/緑色
C/水色
R/赤色
M/紫色
Y/黄色
W/白色
表示属性
[X] S/スクロールバー　T/タブ位置:4
スクロールバーの表示を行う
タブサイズ

● S/サーチパス設定...

次のような各ファイルのサーチパスを設定します.

```
┌──────────────── S/サーチパス設定 ────────────────┐
 サーチパスを指定してください

 X/実行ファイル:              [                        ]
  (.EXE, .COM)

 I/インクルードファイル:      [A:¥INCLUDE ──パス名      ]
  (.BI, .BAS)

 L/ライブラリファイル:        [                        ]
  (.LIB, .QLB)

 E/ヘルプファイル:            [                        ]
  (.HLP)
└──────────────────────────────────────────────────┘
```

● R/マウス機能選択...

マウスの右ボタンをクリックしたときの機能を，次の2つのどちらかに設定します.

```
┌────────── R/マウス機能選択 ──────────┐
 マウス右ボタン機能を設定してください

     (*) C/キーワードヘルプの表示

     ( ) E/クリック行まで実行
├──────────────────────────────────────┤
     【確認】 < 取消 > < H/ヘルプ >
└──────────────────────────────────────┘
```

● Y/構文チェック

Ver.4.2では[E/編集]メニューに入っていましたが，Ver.4.5では[O/オプション]メニューに移されました.

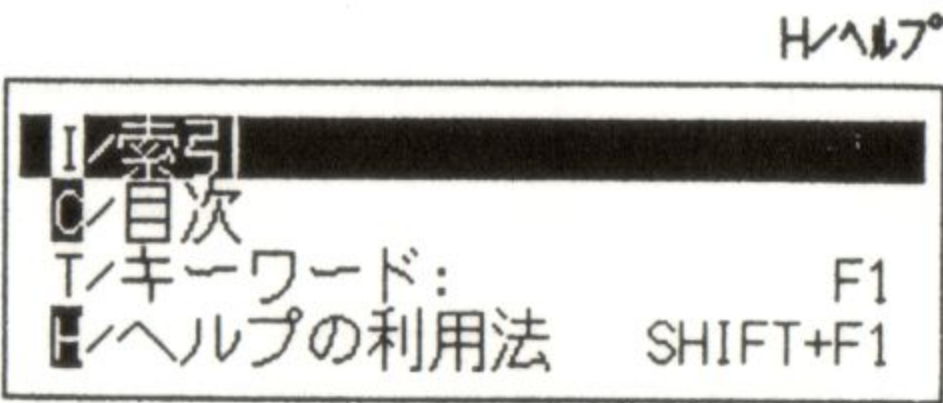
● F/Full メニュー

Easy メニューと Full メニューの切り換えを行います.

■H/ヘルプ (QBアドバイザ)

Ver.4.5では，QBアドバイザと呼ばれる強力なオンラインヘルプをサポートします．

H/ヘルプ

```
I/索引
C/目次
T/キーワード:            F1
H/ヘルプの利用法   SHIFT+F1
```

●I/索引

QBのキーワードをアルファベット順に分類し，選択したキーワードの説明を表示します．

```
                         HELP: 索引
《H/ヘルプの利用法》  《C/項目》  《I/索引》  《P/サポート》
-----------------------------------------------------------------
QuickBASIC のキーワードの一覧を示します．キーワードのヘルプを参照するには，参
照したいキーワードの先頭の1文字を押してから，[F･1] を押すと，キーワードの一覧
が表示されます．参照したいキーワードへカーソルを移動してから再度，[F･1] キー
を押して下さい．
```

A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	R	S	T	U	V	W	X	

⇩ Aを選択し F･1

```
              HELP: 索引-A
ABC                  関数
ACCESS               キーワード
ALIAS                キーワード
AND                  演算子
ANY                  キーワード
APPEND               キーワード
AS                   キーワード
ASC                  関数
ATN Function         関数
```

⇩ ABSを選択

```
                    HELP: ABS 関数:QuickSCREEN
《QuickSCREEN》 《D/詳細》 《E/プログラム例》  《C/目次》   《I/索引》
-----------------------------------------------------------------
ABS - 数式の絶対値を返す数値演算関数です

構文
  ABS(numeric-expression)
```

● **C/目次**

オンラインヘルプの内容の目次を表示します．

```
                          HELP: 目次
《H/ヘルプの利用法》  《C/目次》  《I/索引》  《P/サポート》  《C/著作権》
--------------------------------------------------------------------
《A/QuickBASIC の使い方》

 |―― 《1.ショートカットキーの概要》
 |           《A/編集のためのキー操作》
 |           《B/画面表示のキー操作》
 |           《C/検索のキー操作》
 |           《D/実行とデバッグ操作》
 |           《E/ヘルプキー操作》
 |―― 《2.QuickBASIC の制限事項》
 |―― 《3.以前のバージョンとの違い》
 |―― 《4.QB 起動オプションスイッチ》
 └―― 《5.Quickガイド》

《B/BASIC 言語仕様》

 |―― 《1.目的別キーワード一覧》
 |―― 《2.構文の表記の規則》
 |―― 《3.プログラムの基本的な構成要素》
 |―― 《4.データ型》
 |―― 《5.式と演算子》
 |―― 《6.モジュールとプロシージャ》
 |―― 《7.プログラミング例》
 |―― 《8.文字コード表》
 |―― 《9.キースキャンコード》
 └―― 《0.拡張ライブラリの使い方》
```

● **T/キーワード**

ビューウィンドウのカーソル位置にあるキーワードのヘルプ情報を表示します．

● **H/ヘルプの利用法**

QBのオンラインヘルプの一般的な使い方を表示します．要約すると以下のとおりです．

・ヘルプを参照するには[F･1]を押すか，マウスの右ボタンを押す．
・ヘルプ情報から抜けるには[ESC]を押す．
・ヘルプ画面のスクロールは[ROLL UP]，[ROLL DOWN]を押す．
・《　》で囲まれた項目をハイパーリンクと呼び，この位置にカーソルを移し[F･1]を押すと，その項目のヘルプ情報が表示される．ハイパーリンク間の移動は[TAB]を押す．
・ヘルプ表示したハイパーリンクを20個まで記憶していて，[GRPH]＋[F･1]で前に表示したハイパーリンクの内容を再び表示できる．

7-5 エミュレータ版インタプリタ QBE.EXE

Ver.4.5では，エミュレータ・モードで動作するインタプリタ QBE.EXE をサポートしました.

	代替数値演算	エミュレータ
インタプリタ	QB.EXE	QBE.EXE
ランタイム・ライブラリ	BRUN45A.EXE BRUN45A.LIB	BRUN45E.EXE BRUN45E.LIB
スタンドアロン・ライブラリ	BCOM45A.LIB	BCOM45E.LIB

エミュレータ版において，マシンに数値演算コ・プロセッサがあると，次の演算処理が高速化されます.

- 単精度/倍精度実数の四則演算
- 3角関数
- 対数関数
- 平方根

数値演算コ・プロセッサを搭載していないマシンで，QBE.EXE を使用した場合，数値演算の精度は，コ・プロセッサと同等の高い精度になりますが，演算スピードは QB.EXE より遅くなります.

コンパイルにおいて，代替数値演算とエミュレータ・モードを指定するには次のオプションで指定します.

/FPA　…　代替数値演算
/FPI　…　エミュレータ

なお，QB.EXE からコンパイルすると自動的に/FPA，QBE.EXE からコンパイルすると自動的に/FPI が指定されます.

7-6 言語仕様上の変更点

1 追加されたステートメント/関数

Ver.4.5で追加されたステートメント関数を説明します.

PLAY　（音楽の演奏）

書　式　PLAY　commandstring
　　　　　　　└ミュージックコマンド文字列

機　能　commandstring で示される音符を演奏します.

コマンド	意　　味
On	オクターブの設定. n＝0～6
Nn	音符 n の演奏. n＝0～84
<	1オクターブ上げる
>	1オクターブ下げる
A-G	現在のオクターブで A, B, …, G を演奏. ＋：シャープ, －：フラット
Ln	音符の長さを n 分音符に設定. n＝1～64
Tn	1分間当りの四分音符の数を設定. n＝32～255
MS	各音符を3/4の長さで演奏
MN	各音符を7/8の長さで演奏
ML	各音符を完全な長さで演奏
Pn	n 個の四分音符の休止. n＝1～64
MF	フォアグラウンドで音楽を演奏
MB	バックグラウンドで音楽を演奏
X	X＋VARPTR$(str) で str で示される文字列変数のデータをコマンドとして演奏する

例

```
' ---/* イエスタディワンスモア */---
  テンポ ─┐      ┌── オクターブ
music1$ = "T120 O2 L8 CCD L4 EG L8 GEGEA L4 G"
music2$ = "L8 E L4 E L8 EG L4 AB L8 E L4 G"
PLAY music1$ + music2$
```

R.カーペンター

C

L8 CCD

PLAY （音符の残数の取得） 関数

書式

PLAY(n)
 └ダミー

機能

演奏バッファ(キュー)に残っている音符の数を返します.

例

```
DO WHILE PLAY(0)< >0
LOOP
```

…演奏バッファの音符が残らず演奏されるまで待つ.

ON PLAY （音符残数によるトラッピング）

書式

ON PLAY(limit) GOSUB line
 └分岐する行番号またはラベル
 └分岐限界の音符数(1～32)

機能

BGMキュー(バッファ)の残り音符数がlimitを切ったとき，lineで示されるイベント・トラッピングルーチンへ分岐します.

PLAY ON/OFF/STOP(音符残数トラッピングの制御)

書　式

①PLAY ON
②PLAY OFF
③PLAY STOP

機　能

①ON PLAY によるイベント・トラッピングを可能にします.
②ON PLAY によるイベント・トラッピングを中止します.
③ON PLAY によるイベント・トラッピングを中断します.

SOUND　(音の発生)

書　式

SOUND freq, time
- time ― 音の発生期間
- freq ― 音の周波数(Hz).37～32767

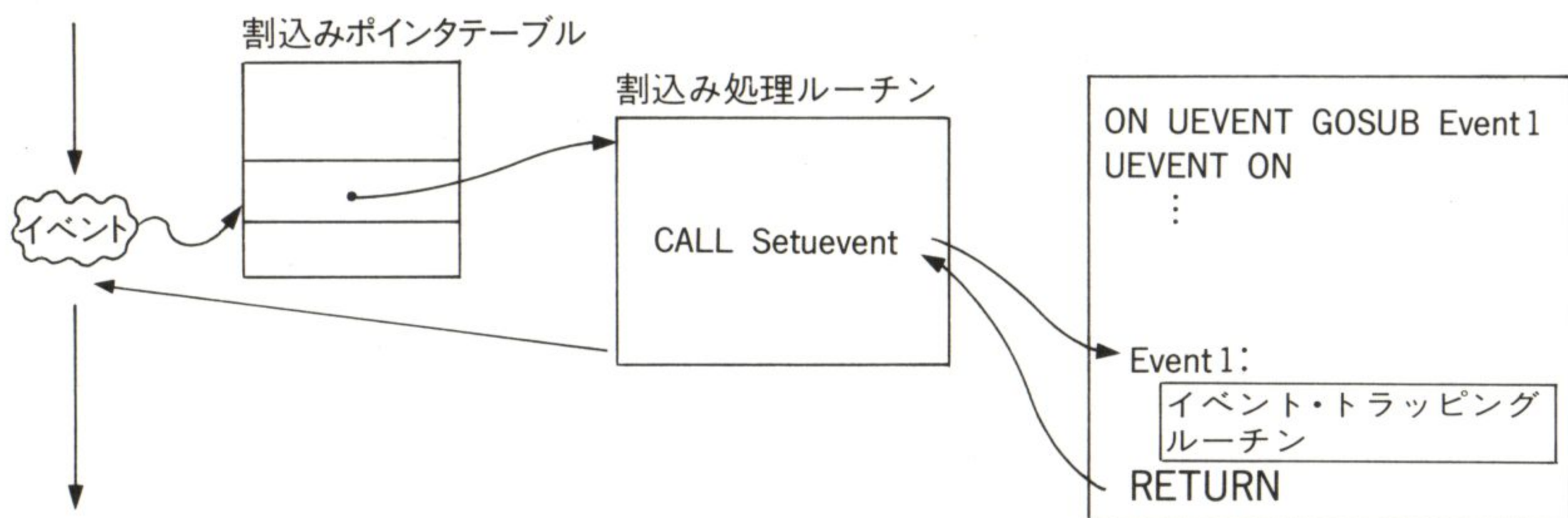
機　能

周波数 freq の音を time で示される期間だけ発生します. time の1単位は約0.055秒です. 1秒を指定するには time に18.2を指定します.
注)PC-9801E/F/M/U では音程を変えることはできません.

ON UEVENT (ユーザ定義イベントによるトラッピング)

書　式

ON UEVENT GOSUB line
- line ― 分岐する行番号またはラベル

機　能

ユーザ定義イベントが起こると，割り込みポインタテーブルに設定されているエントリ・アドレスにしたがって割り込み処理ルーチンへ分岐します．その中でCALL Setueventを行うと，ON UEVENTで設定したイベント・トラッピングルーチンへ分岐します．

割り込みポインタテーブルのエントリ・アドレスの設定と，割り込み処理ルーチンはユーザが行わなければなりません．

Setueventプロシージャは，QB.QLBの中に入っています．

UEVENT ON/OFF/STOP　（ユーザイベント・トラッピングの制御）

書　式

① UEVENT ON
② UEVENT OFF
③ UEVENT STOP

機　能

①ユーザイベント・トラッピングを可能にします．
②ユーザイベント・トラッピングを中止します．
③ユーザイベント・トラッピングを中断します．

SLEEP　（スリープ）

書　式

SLEEP sec
　　　└─スリープする秒

機　能

sec秒だけプログラムの実行を中断(スリープ)します．ただし，スリープ中に何かキーが押されるか，プログラムの実行を再開するイベントが起こると，スリープは解除されます．

secを省略すると，何かキーが押されるか，プログラムの実行を再開するイベントが起こるまでスリープします．

2 変更されたステートメント/関数

Ver.4.5において，機能が変更されたステートメントおよび関数について説明します．

SCREEN　（画面モードの設定）

書式　SCREEN mode, paletteswitch, apage, vpage

機能

mode に82を追加しました．

paletteswitch に 8 色中 8 色のパレットモードを指定する 0 を追加しました．

mode, paletteswitch, apage, vpage に指定できるパラメータは，次のようなマシン群により異なります．

① PC-9801 E/F/M/VM2/VX2/UV2/VM21(VM11,UV21は含まない)
② PC-9801 U
③上記以外

mode	paletteswitch			apage			vpage		
	①	②	③	①	②	③	①	②	③
81	×	×	×	×	×	×	×	×	×
82	×	×	×	×	×	×	×	×	×
84	0	0	0-3	0-3	0-1	0-3	0-1	0-1	0-3
87	0	0	0-3	0-1	0	0-1	0-1	0	0-1
88, 0	0	0	0-3	0-1	0	0-1	0-1	0	0-1

COLOR　（表示色の指定）

書式　COLOR fore, back

機能

スーパーインポーズ・モード(mode = 82, 84, 88, 0)のときの，COLOR 文の第 2 パラメータ back の機能が次のように変更されました．

fore = 0　のときだけ back を次のように指定できます．

0	1	2	3	4	5	6	7
黒	青	緑	水	赤	紫	黄	白

WIDTH　（画面幅の設定）

書式　WIDTH column, row

機能　Ver.4.2ではrowとして25だけでしたが，Ver.4.5では20と25をサポートします．デフォルトは25行モード．

SCREEN　（指定位置の文字または色の取得）　関数

書式　SCREEN(row, column[, colorflag])

機能　Ver.4.2では，SCREENで文字コードを得る場合，漢字の1バイト目と2バイト目を区別できませんでしたが，Ver.4.5では区別できるようになりました．

つまり，指定位置が1バイト目のときは，その文字の漢字連続コードを，2バイト目のときは，負の漢字連続コードを返します．

CSNG$　（2バイト文字を1バイト文字に変換）　関数

書式　CSNG$(str)

機能　Ver.4.5では，全角カタカナの濁音，破裂音，促音の半角への変換と，全角ひらがなの半角への変換をサポートしました．

バ　→　ﾊﾞ
ギャ　→　ｷﾞｬ
あ　→　ｱ

3 環境変数 QBGRPNEC の導入

```
A>SET QBGRPNEC = YES
```

などと，環境変数 QBGRPNEC を定義しておくと，パレットの順番を PC-9801のオリジナルパレットとコンパチブルにします．
ただし，テキスト画面の色は PC-9801とコンパチとはなりません．

0	1	2	3	4	5	6	7
黒	青	赤	紫	緑	黄	水	白

また，QBGRPNEC が定義されていると，次のような動作をとります．
- プログラム実行/終了時にグラフィック画面のクリアを行わない．
- プログラム実行/終了時にパレットを初期化しない．

4 STOP / HELP キーによるプログラム割り込みのサポート

STOP キーでの割り込みを，

```
ON KEY(30) GOSUB line
```

HELP キーでの割り込みを，

```
ON KEY(31) GOSUB line
```

で行えるようにしました．

5 ソーステキスト，ライブラリの互換性

標準ファイルでセーブしたソーステキストは，Ver.4.2のものを Ver.4.5でも読めますが，Ver.4.5のものを Ver.4.2で読むことはできません．テキスト・ファイルの形式で保存してあるものはどちらでも読めます．
Ver.4.2で作成したライブラリを，Ver.4.5で利用できませんし，その逆もできません．利用するバージョンに合わせて，ライブラリを再構築してください．

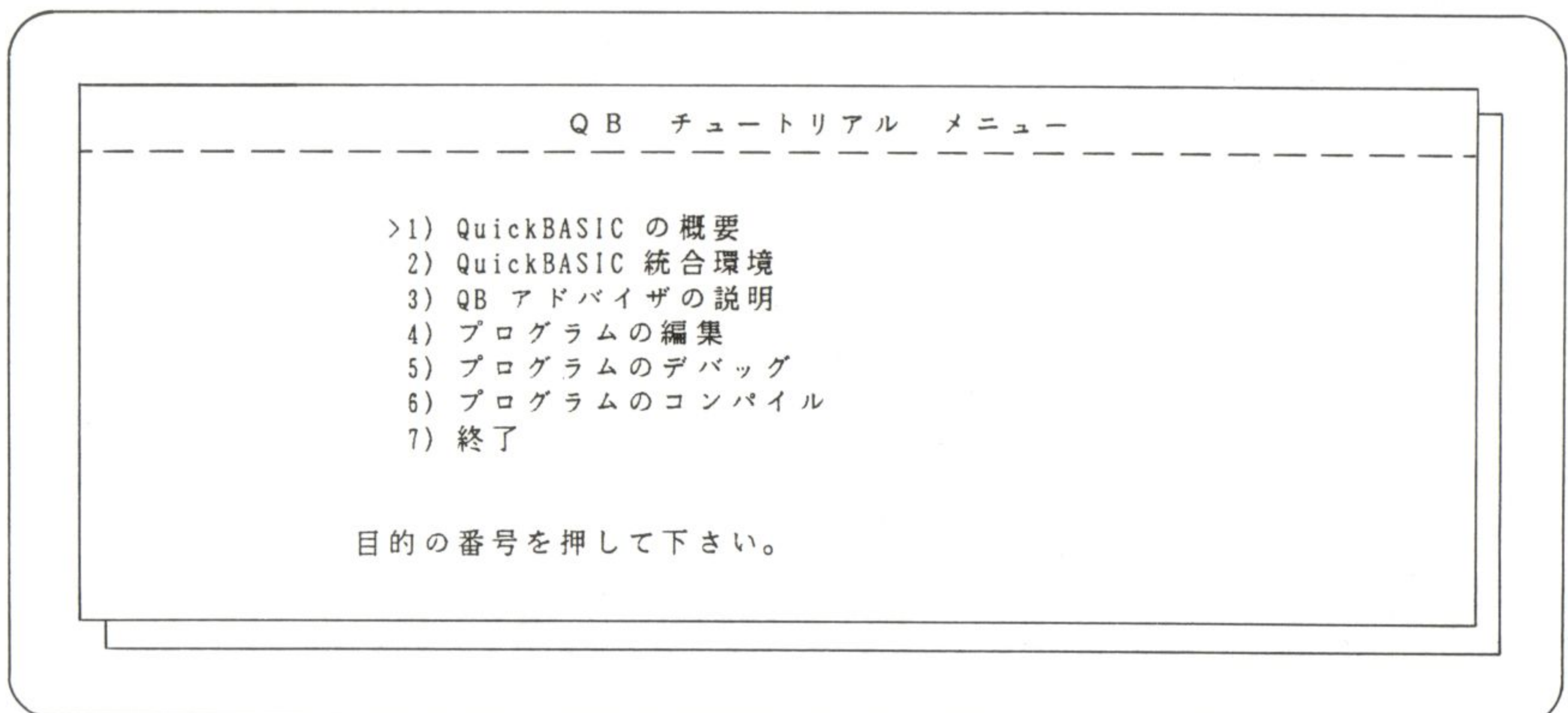
7-7 QB チュートリアル

Ver.4.5では，QB の概要を画面上で学習させるソフトとしてQB チュートリアルを提供しました．

セットアップディスクをカレントドライブに置き，サブディレクトリ LEARN をカレントディレクトリにして，

```
A>LEARN
```

とすることで，次のような QB チュートリアルを起動できます．

```
                    Q B   チュートリアル   メニュー
------------------------------------------------------------

            >1) QuickBASIC の概要
             2) QuickBASIC 統合環境
             3) QB アドバイザの説明
             4) プログラムの編集
             5) プログラムのデバッグ
             6) プログラムのコンパイル
             7) 終了


          目的の番号を押して下さい。
```

なお，SETUP ユーティリティを起動し，そこから QB チュートリアルに入ることもできます．

7−8 HELPMAKE

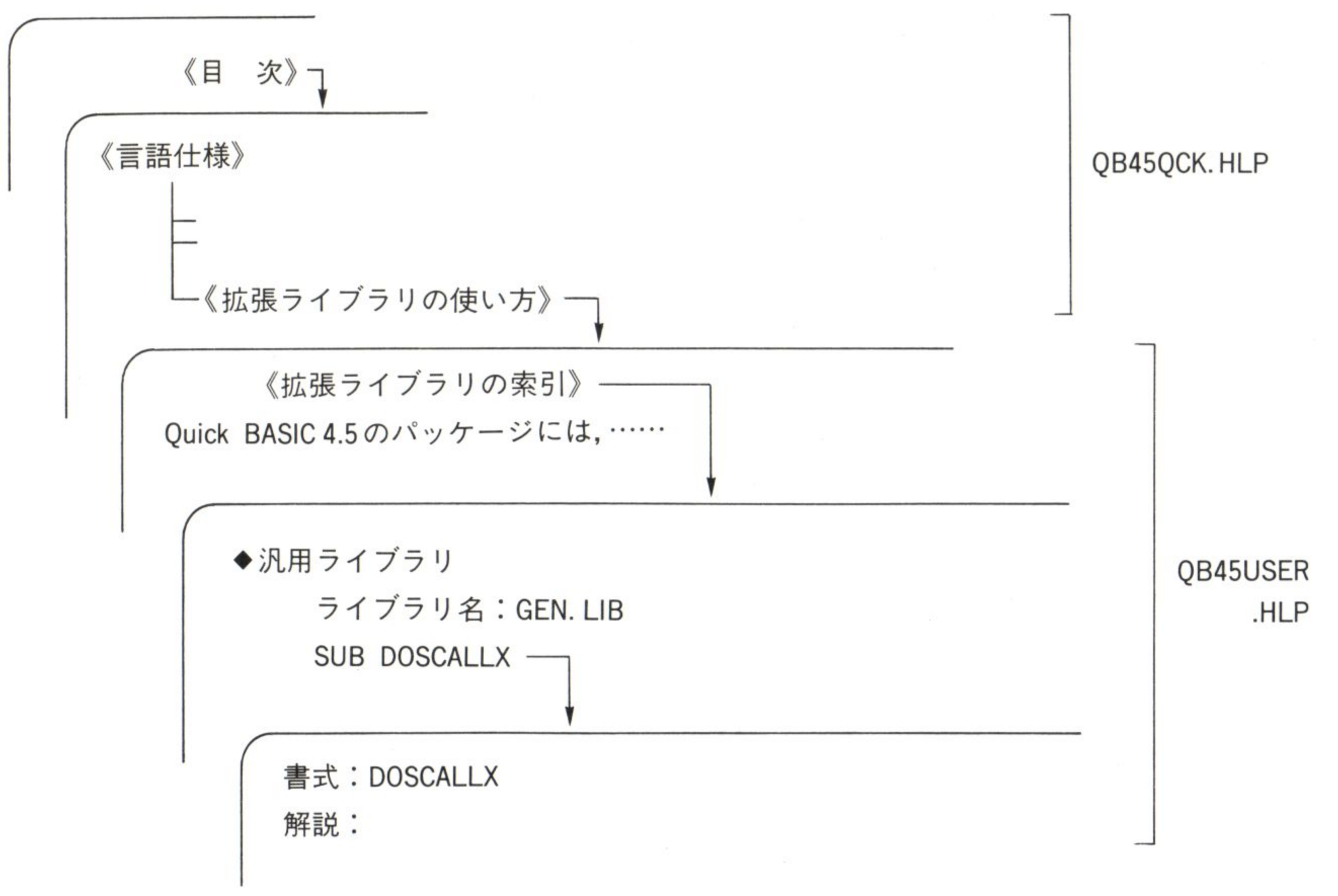
1 ヘルプファイルの概要

HELPMAKEは，ヘルプファイルを作成するためのユーティリティです．
QBが標準で使うヘルプファイルは次の4つです．

QB45QCK.HLP　… 索引，目次，関数，ステートメントのトピックが記述されている．
QB45ENER.HLP　… メニューやダイアログボックスの解説，エラーの解説
QB45ADVR.HLP　… 関数，ステートメントの詳しい解説やプログラム例
QB45USER.HLP　… ユーザライブラリの説明

さて，このヘルプファイルは，ファイルサイズを小さくするために圧縮(パック)されていますので，エディタで直接編集することはできません．ヘルプファイルの内容を変更したい場合は，HELPMAKEを用いて一度テキスト・ファイルに変換し，必要な修正を行ったあと，再びHELPMAKEを用いて，逆にテキスト・ファイルをヘルプファイルに変換します．
次の図は，拡張ライブラリのDOSCALLXのヘルプを見る過程を示したものです．

《目　次》
《言語仕様》
《拡張ライブラリの使い方》
QB45QCK.HLP

《拡張ライブラリの索引》
Quick BASIC 4.5のパッケージには，……
◆汎用ライブラリ
ライブラリ名：GEN. LIB
SUB DOSCALLX
書式：DOSCALLX
解説：
QB45USER.HLP

《　》の中とSUB DOSCALLX(緑色で表示される)はハイパーリンクになっています.

さて，これらのヘルプ参照はハイパーリンクにより結合されているわけですが，その構造を以下に示します.

QB45QCK.HLPをテキスト・ファイルに変換してスクリーンに表示してみましょう.

```
A>helpmake /d b:qb45qck.hlp
Microsoft (R) Help File Maintenance Utility
Version 1.00
Copyright (c) Microsoft Corp 1988-1989. All rights reserved.
Decoding: b:qb45qck.hlp
      .
      .
      .
¥i 《¥a¥pB/BASIC 言語仕様¥i》 ¥vqb45advr.hlp!.kemeny¥v¥p

   ├─ ¥i 《¥a¥p1.目的別キーワード一覧¥i》¥vqb45advr.hlp!.fk¥v¥p
   ├─ ¥i 《¥a¥p2.構文の表記の規則¥i》 ¥vqb45advr.hlp!.tok¥v¥p
   ├─ ¥i 《¥a¥p3.プログラムの基本的な構成要素¥i》 ¥vqb45advr.hlp!.fun¥v¥p
   ├─ ¥i 《¥a¥p4.データ型¥i》 ¥vqb45advr.hlp!.dtp¥v¥p
   ├─ ¥i 《¥a¥p5.式と演算子¥i》 ¥vqb45advr.hlp!.exo¥v¥p
   ├─ ¥i 《¥a¥p6.モジュールとプロシージャ¥i》 ¥vqb45advr.hlp!.mxp¥v¥p
   ├─ ¥i 《¥a¥p7.プログラミング例¥i》 ¥vqb45advr.hlp!.pswitch¥v¥p
   ├─ ¥i 《¥a¥p8.文字コード表¥i》 ¥vqb45advr.hlp!.ac¥v¥p
   ├─ ¥i 《¥a¥p9.キースキャンコード¥i》 ¥vqb45advr.hlp!.kbsct¥v¥p
   └─ ¥i 《¥a¥p0.拡張ライブラリの使い方¥i》 ¥vqb45user.hlp!.userlibh¥v¥p
                                              ↑            ↑── ハイパーリンクID
                                              └── ヘルプファイル名
```

QB45USER.HLPのテキスト・ファイルは，QB45USER.QHとして提供されていますので，TYPEで表示してみましょう.

```
A>type qb45user.qh
.context .userlibh ←――― ハイパーリンク ID
.context @userlibh
:n拡張ライブラリの概要
¥i 《¥i¥aC/目次¥i》 ¥i¥v-9996¥v¥p ¥i《¥i¥al索引¥i》 ¥i¥v-9997¥v¥p ¥i《¥i¥aU/拡張
ライブラリの索引¥i》 ¥iv@userlibc¥v¥p
       .   ↖― ハイパーリンク   ↖――― ハイパーリンク ID
       .
       .
.context .userlibc
.context @userlibc ←――― ハイパーリンク ID
:n拡張ライブラリの索引

◆  汎用ライブラリ
      ライブラリ名          : GEN.LIB
      クイックライブラリ名: GEN.QLB
      インクルードファイル: QB.BI, GEN.BI

      ¥i¥aSUB      DOSCALLX¥i¥v@DOSCALL¥v¥p                INT 21Hを呼び出しま
す.  DS/ES対応 ↖― ハイパーリンク ↖――― ハイパーリンク ID
      ¥i¥aSUB      DOSCALL¥i¥v@DOSCALL¥v¥p                 INT 21Hを呼び出しま
す.
       .
       .
       .
.context @DOSCALL ←――― ハイパーリンク ID
.context @DOSCALLX
:nDOSCALL/DOSCALLX SUB PROGRAM
DOSCALL/DOSCALLX の利用法

ライブラリ名          : GEN.LIB
クイックライブラリ名: GEN.QLB
インクルードファイル: QB.BI, GEN.BI

構文: DOSCALL[X] inreg, outreg
```

ヘルプテキストの構成の概要を以下に示します.

```
.context  ハイパーリンク ID
:n 見出し
    説明文
    ⋮
        ¥a  ハイパーリンク  ¥v  ハイパーリンクID
```

各説明文は, .context で始まり, 後の〈ハイパーリンク ID〉がタイトルの役割をし, 他の説明文中のハイパーリンクからの参照を可能にします. :n の後の〈見出し〉はヘルプ画面に表示されたときの見出しになります.

説明文中にハイパーリンクを設定するには, ハイパーリンクにする項目を¥a と¥v で囲み, そのあとにそのハイパーリンクへの〈ハイパーリンク ID〉を書きます.

2 ヘルプファイルの変更例

ユーザライブラリヘルプ(QB45USER.HLP)に新しい説明文を追加する具体例を以下に示します.

● QB45USER.QH(テキスト・ファイル)

```
◆  グラフィックライブラリ

    ライブラリ名          : GRAPH.LIB
    クイックライブラリ名  : GRAPH.QLB
    インクルードファイル  : QB.BI, GRAPH.BI

   ¥i¥aSUB     GROLL¥i¥v@GROLL¥v¥p                        グラフィック画面をス
   ¥i¥aSUB     KPUT¥i¥v@KPUT¥v¥p                         漢字を描画します.
   ¥i¥aSUB     LINEXOR¥i¥v@LINEXOR¥v¥p                      XOR モードで線を引

*  プリンタライブラリ

   ライブラリ名         : PRINTER.QLB

   ¥i¥aSUB     LPUTC¥i¥v@LPUTC¥v¥p                  プリンタへの1文字出力
      .
      .
      .

.context @LPUTC
:nPRINTER SUB PROGRAM

書式 : LPUTC c%

解説 : c%で示される文字をプリンタに出力します.
```

追加した説明文（プリンタへの1文字出力の行）

ハイパーリンク（SUB）／ハイパーリンクID（@LPUTC）

.context @LPUTC ← ハイパーリンクID

追加した説明文（:nPRINTER SUB PROGRAM 〜 解説の行）

修正したヘルプテキスト・ファイル(QB45USER.QH)を, ヘルプファイル(QB45 USER. HLP)に変換する様子を以下に示します.

制御文字（必ず/a：と指定）

ファイル圧縮率を最大（約50%）

出力ファイル

ヘルプファイルの1行の最大長

テキスト・ファイル

```
A>helpmake /a: /e /ob:qb45user.hlp /w150 qb45user.qh
Microsoft (R) Help File Maintenance Utility
Version 1.00
Copyright (c) Microsoft Corp 1988-1989. All rights reserved.
```

3 HELPMAKEの仕様概要

● HELPMAKE の起動

A>HELPMAKE　[[option]　{/En　| /D}　[option]]　sourcefile

●オプション

		意　味
エンコードオプション	/Ac	ヘルプデータベースの制御文字としてcを指定．QBではつねに/A:と指定
	/En	ファイル圧縮率．nは0〜15．nを省略すると最高の圧縮率（約50%）
	/H	HELPMAKEのヘルプ
	/L	作成するヘルプファイルがデコード（アンパック）されないようにロックする
	/Odestfile	出力するヘルプファイル名
	/S2	QB用オプションで省略できる
	/Vn	HELPMAKEの処理過程の表示方法のレベルをn（0〜6）で指定．nを省略すると全過程の表示
	/Wn	ヘルプファイルの1行の長さをn（11〜255）で指定
デコードオプション	/D	ヘルプファイルのデコード
	/DU	デコードしたテキストファイル中のクロスリファレンス，表示指定情報を削除
	/H	HELPMAKEのヘルプ
	/Odestfile	作成するテキスト・ファイル名．省略するとスクリーンに出力
	/Vn	HELPMAKEの処理過程の表示方法のレベルをn（0〜3）で指定．nを省略すると全過程の表示

●特別な context 名

context	説　明
h.default	デフォルトのヘルプ画面
h.notfound	要求されたcontextが見つからないときに表示するトピック
h.pg1	論理的に最初にあるトピック
h.pg$	論理的に最後にあるトピック
h.	ヘルプ項の参照先であることを示す
m.	メニュー項目のヘルプの参照先であることを示す
e.	エラーのヘルプ参照先であることを示す

●テキストファイル中の制御文字

制御文字	意　味
:In	トピックを表示するウィンドウの行数を n で指定
:n text	ヘルプのタイトルを text で指定
¥a	ハイパーリンクの先頭
¥b, ¥B	太字指定のオン/オフ
¥i, ¥I	イタリック指定のオン/オフ
¥p, ¥P	すべての属性をオフ
¥u, ¥U	下線指定のオン/オフ
¥v, ¥V	ハイパーリンクの終わり
¥¥	¥マーク自身

7-9 ユーザライブラリ

Ver.4.5では次のようなユーザライブラリが提供されています．

GEN.QLB(GEN.LIB)　　　　…汎用ライブラリ
GRAPH.QLB(GRAPH.LIB)　　…グラフィックライブラリ
MOUSE.QLB(MOUSE.LIB)　　…マウスライブラリ

これらのライブラリは，～．QLB（クイックライブラリ），～．LIB（ライブラリ），～．BI(プロシージャ宣言ファイル)，～．BAS（ソーステキスト)の形態で提供されています．使用にあたってはライブラリをLIBディレクトリ，インクルードファイルをINCLUDEディレクトリに置き，たとえばGRAPH.QLBを使用するには，

```
A>QB /LGRAPH↵
```

と起動します． ユーザプログラムでは， QB.BI, GRAPH.BIをインクルードします．

```
'$INCLUDE: 'qb.bi'
'$INCLUDE: 'graph.bi'

SCREEN 0: CLS
KPUT 100, 100, "日本語のグラフィック画面への表示", 1, 0
LINE (90, 95)-(370, 120), 5, B
```

日本語のグラフィック画面への表示

各ライブラリの仕様ドキュメントは，ソーステキスト(GEN．BAS，GRAPH．BAS，MOUSE．BAS)に記載されています．
以下にその仕様の要約を説明します．

1 汎用ライブラリ

GEN.QLB のライブラリについて説明します．このライブラリを使用するユーザプログラムでは，QB.BI と GEN.BI をインクルードしてください．

Fep　（FEP の ON/OFF）

書式

```
Fep f$
    └─"ON"または"OFF"
```

機能

FEP を ON または OFF します．

GetCommand$　（コマンドライン引数の取得）　関数

書式

```
GetCommand$(n%)
            └─引数の番号
```

機能

n%番目のコマンドライン引数を取得します．

GetFile$　（ディレクトリ・エントリの検索）　関数

書式

```
GetFile$ (FileName$, Attr%, DtaData AS DtaType, Func%)
          ↑          ↑      ↑                   └─0,非0
          │          │      └─ディレクトリ・エントリ用バッファ
          │          └─ファイル属性
          └─検索するファイル名．ワイルドカードを含む
```

機能

FileName$で示されるファイルのディレクトリ・エントリを検索し，DtaData に返します．

GetFile$の戻り値は取得されたファイル名です．

```
TYPE DtaType
    DtaDummy AS STRING * 21
    DtaAttr  AS STRING * 1
    DtaTime  AS STRING * 2
    DtaDate  AS STRING * 2
    DtaSize  AS LONG
    DtaName  AS STRING * 14
END  TYPE
```

Attr%には検索されたファイルの属性が次のような値で返されます．

0	通常ファイル
1	リードオンリー・ファイル
2	隠しファイル
4	システム属性
8	ボリューム名
&H10	ディレクトリ
&H20	アーカイブ

ワイルド・カードを含むディレクトリ・エントリの検索では，GetFile$を続けて呼び出すことになりますが，一番最初の呼び出しでは，Func%を0に指定し，2度目以後は非0を指定します．2度目以後の呼び出しではFileName$の文字列は意味を持ちません．

例

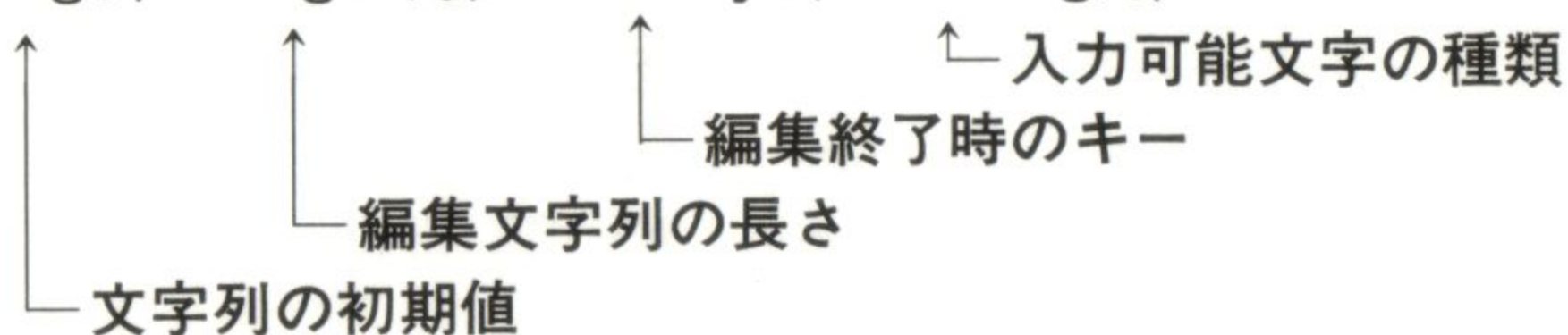
```
' ---/* ディレクトリ・エントリの検索 */---

'$INCLUDE: 'qb.bi'
'$INCLUDE: 'gen.bi'

DIM buf AS DtaType

INPUT "File Name "; f$

a$ = GetFile$(f$, Attr%, buf, 0)
DO WHILE a$ <> ""
    PRINT a$
    a$ = GetFile$("", Attr%, buf, 1)
LOOP
```

```
File Name ? *.hlp
QB45ADVR.HLP
QB45ENER.HLP
QB45QCK.HLP
QB45USER.HLP
```

Edit$ （文字列の編集） 関数

書式

```
Edit$(Arg$, Length%, Lastkey$, StrFlag%)
```

- Arg$ … 文字列の初期値
- Length% … 編集文字列の長さ
- Lastkey$ … 編集終了時のキー
- StrFlag% … 入力可能文字の種類

機能

Arg$の文字列が画面に表示され，その文字列を編集した結果が返されます．編集範囲は，Arg$の文字列を含めてLength%で指定した長さです．[ESC]，[↵]などのキーにより編集を終了しますが，このときの文字がLastkey$に返されます．

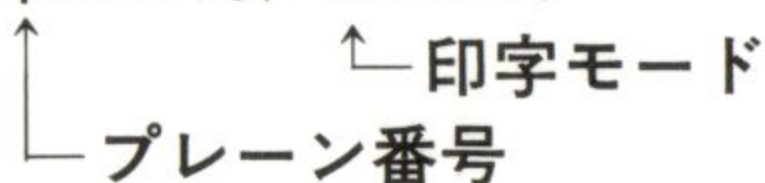
StrFlag%は，入力可能文字の種類で以下の値を指定します．

< -1	半角
= -1	FEP ON にして半角
= 0	すべての文字
= 1	全角
> 1	FEP ON にして全角

HCOPY （画面のハードコピー）

書式

```
HCOPY plane%, mode%
```

- plane% … プレーン番号
- mode% … 印字モード

機能

テキスト画面とグラフィック画面のハードコピーをとります．

plane%に指定する値は，0～2のビットが印字するプレーンを示します．つまり第0ビット…Blueプレーン，第1ビット…Redプレーン，第2ビット…Greenプレーンとなります．

mode%が0ならノーマル，1なら縮小印字となります．

2 グラフィック・ライブラリ

GRAPH．QLBのライブラリについて説明します．このライブラリを使用するユーザプログラムでは，QB．BIとGRAPH．BIをインクルードしてください．

GROLL　（グラフィック画面のスクロール）

書式

```
GROLL x%, y%
      │    └y方向スクロールドット数
      └x方向スクロールドット数
```

機能

x%はx方向のスクロールドット数で8ドットの倍数を指定します．正の値なら右へ，負なら左へスクロールします．

y%はy方向のスクロールドット数で1ドットの単位で指定します．正の値なら下へ，負なら上へスクロールします．

KPUT　（文字列のグラフィック画面への表示）

書式

```
KPUT x%, y%, StrArg$, fore%, back%
     └──┬──┘   │        │      └バックグラウンド色
        │      │        └フォアグラウンド色
        │      └表示する文字列
        └表示位置(左上座標)
```

機能

StrArg$で示される文字列を，グラフィック画面の(x%，y%)位置に表示します．

LINEXOR（XORモードでの直線）

書式

```
LINEXOR x1%, y1%, x2%, y2%, plane%, style%
        └───┬───┘  └───┬───┘   │       └ラインスタイル
            │          │       └描画プレーン    (0～&HFFFF)
            └始点       └終点
```

機能

描画画面との排他的論理和(XOR)をとり，その結果にしたがって線を引きます．描画は1プレーンのみで，複数プレーンへは同時に描画できません．

plane% = 0 … 青プレーン
　　　　 1 … 緑プレーン
　　　　 2 … 赤プレーン

3 マウスライブラリ

MOUSE．QLB のライブラリについて説明します．このライブラリを使用するユーザプログラムでは，QB．BI と MOUSE．BI をインクルードしてください．

MouseBorder （マウス移動範囲の設定）

書　式　MouseBorder row1%，col1%，row2%，col2%

row1%，col1%：左上座標
row2%，col2%：右下座標

機　能　マウス・カーソルの移動範囲を(row1%，col1%)－(row2%，col2%)に設定します．

MouseHide （マウスを非表示）

書　式　MouseHide

機　能　マウス・カーソルを表示しません．

MouseInit （マウスの初期化）

書　式　MouseInit

機　能　マウスの初期化を行います．マウスプロシージャの実行前に必ず一度実行しておく必要があります．

MouseInit 直後はマウスは非表示状態です．

MouseLocate （マウス位置の設定）

書　式　MouseLocate row%，col%

row%，col%：マウス位置の座標

機　能　マウス・カーソルの位置を(row%，col%)に設定します．

MouseMode　（マウス・タイプの指定）

書式

```
MouseMode Func%
          └─マウス・タイプ
```

機能

マウス・カーソルのタイプをFunc%で指定します．

0 …　グラフィック・カーソル
非0 …　ソフトウェア・カーソル

MousePoll　（マウスの状態の取得）

書式

```
MousePoll row%, col%, lbutton%, rbutton%
                                  └─右ボタンの状態
                         └─左ボタンの状態
          └─マウスの現在座標 (row%, col%)
```

機能

マウスの位置，ボタンの状態を取得します．得られたボタン状態が0ならボタンOFF，−1ならボタンONを意味します．

例

次のプログラムは，左ボタンを押したときの2点間を直線で結ぶものです．

右ボタンを押せばプログラムから抜けます．

マウスの「ボタンを押す(離す)」という動作は，コンピュータのスピードに比べてはるかに遅いので，もしボタン押し下げの検知部分がループ内にある場合には，ボタンを1回押したつもりでも，何回も押したことになってしまいます．そこでフラグfreeを用い，ボタンが離されるとfreeを1にセットするようにしておき，ボタン押し下げ検知部分ではfreeが1でなければボタンがいくら押されていても押したとみなさないようにします．

```
' ---/* 2点間の直線を引く */---

'$INCLUDE: 'qb.bi'
'$INCLUDE: 'mouse.bi'

SCREEN 0: CLS
flag = 0: free = 1
                    ← 始点/終点を表すフラグ
Mouseinit
MouseLocate 200, 320
MouseShow

MousePoll y%, x%, Left%, Right%
DO WHILE Right% <> -1 ←――― 右ボタンが押されるまで
    IF Left% = -1 AND free = 1 THEN
        IF flag = 0 THEN
            xs% = x%
            ys% = y%          ← 始点のときの処理
            flag = 1
        ELSE
            MouseHide
            LINE (xs%, ys%)-(x%, y%), 1
            flag = 0              ← 終点のときの処理
            MouseShow
        END IF
        free = 0
    END IF
    IF Left% = 0 THEN
        free = 1                  ← 左ボタンが離されたときの処理
    END IF
    MousePoll y%, x%, Left%, Right%
LOOP
MouseHide
```

MousePush（マウス・ボタン ON 時の状態の取得）

書　式

MousePush button%, row%, col%, lbutton%, rbutton%

- button% ― ONボタンの指定(0:左ボタン 1:右ボタン)
- row%, col% ― マウスの現在座標
- lbutton% ― 左ボタンの状態
- rbutton% ― 右ボタンの状態

機　能

マウスの位置，ボタンの状態を取得します．得られたボタン状態が0ならボタンOFF，−1ならボタンONを意味します．

なお，button%には，最後にこのプロシージャが呼ばれたときから指定された(呼び出し時のbutton%で指定した)ボタンの押された回数が返されます．

MouseReady （マウスが利用可能か調べる） 関数

書式

MouseReady

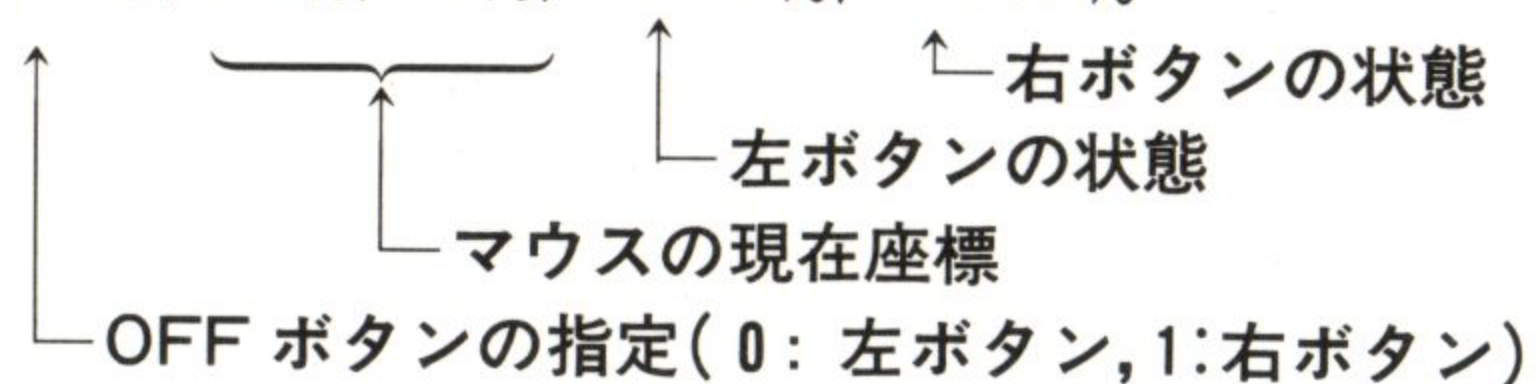
機能

0 … マウスが使える
−1 … マウスは使えない

MouseRelease（マウス・ボタンOFF時の状態の取得）

書式

```
MouseRelease button%, row%, col%, lbutton%, rbutton%
```

- button% … OFFボタンの指定（0：左ボタン，1：右ボタン）
- row%, col% … マウスの現在座標
- lbutton% … 左ボタンの状態
- rbutton% … 右ボタンの状態

機能

ボタンの状態を取得します．得られたボタン状態が0ならボタンOFF，−1ならボタンONを意味します．
なお，button%には，最後にこのプロシージャが呼ばれたときから指定された(呼び出し時のbutton%で指定した)ボタンの離された回数が返されます．

MouseShow （カーソルの表示）

書式

MouseShow

機能

マウス・カーソルを表示します．

●記号／英数字項目

■A

■B

■C

■D

■E

F

G

H／I／J

K

L

■M

■N

■O

■P

■Q

■R

■S

■T

■U／V／W／X

●和文項目

■ア

■カ

■サ

■タ

■ナ

■ハ

■マ

■ヤ／ラ／ワ

河西朝雄（かさい　あさお）

昭和49年　山梨大学工学部電子工学科卒
現　　在　長野県岡谷工業高等学校情報技術科教諭
　　　　　第一種情報処理技術者
著　　者　「Cプログラミング技法」，「Cプログラミングノート」，「Cプログラミング入門」，「MS-DOSシステムコールハンドブック」，「TURBO PASCALハンディ・マニュアル」（ナツメ社）「構造化BASIC」，「MS-DOS実用マクロアセンブラ」，「TURBO C初級プログラミング上・下」，「最新はじめてのBASIC」，「最新はじめてのTURBO C」（技術評論社）など

Ver.4.2～4.5
QuickBASIC初級プログラミング入門（上）

平成２年４月30日　初版　第１刷発行
平成５年12月25日　初版　第９刷発行

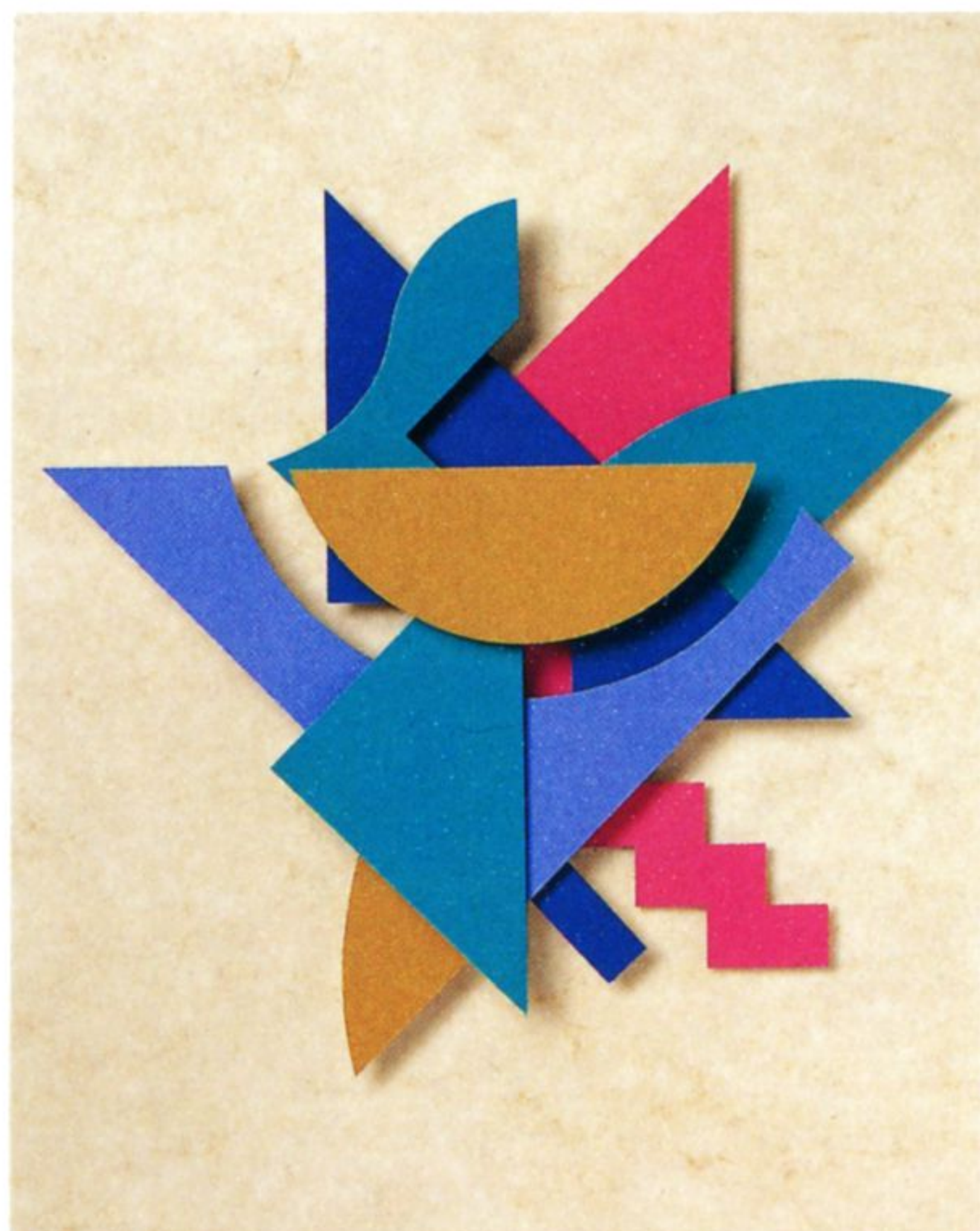
著　者　河西朝雄
発行者　片岡　巖
発行所　株式会社技術評論社
　　　　東京都新宿区愛住町８番地８
　　　　電話　03(3225)2300　営業部
　　　　　　　03(3225)3293　編集部
印刷／製本　加藤文明社

定価はカバーに表示してあります

ISBN4-87408-353-6 C3055

Printed in Japan

M-cord 210041

Ver.4.2~4.5
Quick BASIC
初級プログラミング入門[上]
河西朝雄[著]

ISBN4-87408-353-6 C3055 P2100E 定価2100円（本体2039円）